

AD-A238 830



**JOINT DATA SYSTEMS
SUPPORT CENTER**

**USERS MANUAL
UM 7-91
1 FEBRUARY 1991**

USERS MANUAL FOR THE GRAPHIC INFORMATION PRESENTATION SYSTEM (GIPSY)

**APPROVED FOR
PUBLIC RELEASE
DISTRIBUTION UNLIMITED**

REPRODUCED BY
U.S. DEPARTMENT OF COMMERCE
NATIONAL TECHNICAL
INFORMATION SERVICE
SPRINGFIELD, VA 22161



07

8

27

07-018

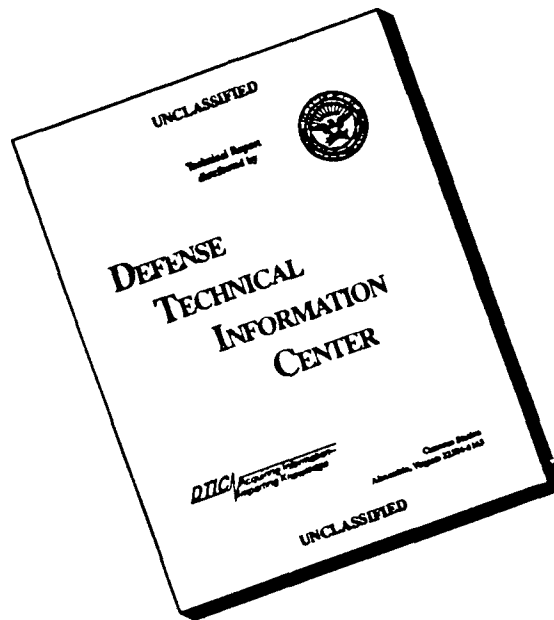
RECORD OF CHANGES

Change Number	Dated	Date Entered	Signature of Person Making Change

DCA FORM 65

MAR 87

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.



DEFENSE INFORMATION SYSTEMS AGENCY
DEFENSE SYSTEMS SUPPORT ORGANIZATION
WASHINGTON, D.C. 20301-7010

IN REPLY JNKG
REFER TO

TO: RECIPIENTS

SUBJECT: Change 1 to GIPSY Users Manual, UM 7-91

1. This is change 1 to JDSSC Users Manual UM 7-91 titled Graphic Information Presentation System (GIPSY) Users Manual (UM 7-91), dated 1 February 1991. Remove obsolete pages and destroy them in accordance with applicable security regulations and insert the new pages as indicated below:

Remove Pages

iv through vii
1-5 through 1-8
2-3 through 2-6
3-17 through 3-18
3-23 through 3-24
3-29 through 3-38
3-43 through 3-44
3-47 through 3-48
H-1 through H-10
I-1 through I-8

Insert New Pages

iv through vii
1-5 through 1-8.2
2-3 through 2-6.2
3-17 through 3-18
3-23 through 3-24.2
3-29 through 3-38
3-33 through 3-44
3-47 through 3-48
H-1 through H-10
I-1 through I-8

2. The effective date of these change pages is 6 May 1992.

3. When these changes have been made, post an entry in the Record of Changes for the manual and file this letter before the title page.

36 Enclosures a/s

Thomas R. Epperson
THOMAS R. EPPERSON
Deputy Director, NMCS ADP

ERRATA AD A 228250

JOINT DATA SYSTEMS SUPPORT CENTER

Users Manual UM 7-91

1 February 1991

USERS MANUAL
FOR THE
GRAPHIC INFORMATION PRESENTATION SYSTEM (GIPSY)

SUBMITTED BY:

for *David W. Hall*
Jeanne L. Muenzen
Chief, Information
Systems Branch

APPROVED BY:

Thomas R. Epperson
Thomas R. Epperson
Deputy Director
NMCS ADP Directorate

Copies of this document may be obtained from the Defense Technical Information Center, Cameron Station, Alexandria, Virginia 22304-6145.

Approved for public release; distribution unlimited.

ACKNOWLEDGMENT

This document was prepared under the general direction of the Chief, Information Systems Branch (JNGG); Chief, General Applications Division (JNG); and the Deputy Director, National Military Command System (NMCS) Automated Data Processing (ADP) Directorate (JN).

CONTENTS

Section	Page
ACKNOWLEDGMENT	ii
ABSTRACT	xi
1. GENERAL	1-1
1.1 Purpose of the Users Manual	1-1
1.1.1 Project Orientation	1-1
1.1.1.1 Introduction to Graphics	1-1
1.1.1.2 Purpose of Project	1-2
1.1.1.3 Scope of Project	1-2
1.1.1.4 Audience	1-2
1.1.1.5 Essential Considerations	1-2
1.2 Project References	1-5
1.3 Terms and Abbreviations	1-6
1.4 Security and Privacy	1-9
2. SYSTEM SUMMARY	2-1
2.1 System Overview	2-1
2.2 System Operation	2-3
2.3 System Configuration	2-3
2.4 System Organization	2-5
2.5 System Performance	2-5
2.6 Contingencies and Alternate Modes of Operation	2-5
2.7 Database/Data Bank	2-5
2.8 General Description of Inputs, Processing, Outputs	2-6
2.8.1 Input and Processing Operations	2-6
2.8.2 Language Overview	2-9
2.8.3 Outputs	2-12
3. STAFF FUNCTIONS RELATED TO TECHNICAL OPERATIONS	3-1
3.1 Initiation Procedure	3-1
3.2 GIPSY Language: Syntax and Semantics	3-1
3.2.1 Composition Rules	3-3
3.2.2 GIPSY Command Options	3-6
3.2.3 Classification, Title, and Auxiliary Functions	3-9
3.2.3.1 Classification Marking	3-9
3.2.3.2 Report Titles	3-11
3.2.3.2.1 Modifying Title Lines	3-11
3.2.3.3 Language Input	3-11
3.2.3.4 Changing Language Input Modes	3-14
3.2.3.5 Operating Environment Attributes	3-14
3.2.3.5.1 Define Terminal Command	3-16
3.2.3.6 Saving GIPSY Data Structures	3-18
3.2.3.6.1 Saving the File Descriptor Table (FDT)	3-18
3.2.3.6.2 Saving the Process Control Statements (FCS)	3-19
3.2.3.6.3 Saving the Graphic Data Set (GDS)/Report	3-19

Section	Page
3.2.3.6.4	Saving the Qualified Data File (QDF) 3-19
3.2.3.6.5	Saving the Qualified Descriptor Table (QDT) 3-20
3.2.3.6.6	Saving the Directive Action and File Control (DAFC) File 3-20
3.2.3.6.7	Saving the QDF or GDS in the Data Interface Format (DIF) 3-22
3.2.3.7	Saving Data Fields 3-22
3.2.3.8	Interrupt Statements 3-23
3.2.3.9	Comments 3-26
3.2.3.10	Executing Other TSS Commands 3-26
3.2.3.11	Clearing Data Structures 3-27
3.2.3.12	Executing a User Program 3-27
3.2.3.13	Terminating the GIPSY Session 3-27
3.2.3.14	GIPSY Color Processing 3-27
3.2.3.15	User Defined Subroutines 3-28
3.2.3.16	JDAC Command 3-28
3.2.4	User Data Input 3-28
3.2.4.1	The Data File 3-29
3.2.4.2	The Index File 3-29
3.2.5	Describing the Data Records 3-30
3.2.5.1	Describing an I-D-S/II Integrated File 3-30
3.2.5.2	Describing Data File Records to GIPSY 3-30
3.2.5.3	Describing the Index File 3-36
3.2.5.4	Describing GLOBAL and QUALIFY Records 3-37
3.2.6	Conditional Expressions 3-37
3.2.7	Predefining Conditional Expressions 3-41
3.2.8	Data Retrieval 3-43
3.2.9	Arithmetic Expressions 3-45
3.2.10	Predefining Arithmetic Expressions 3-45
3.2.11	Data Modifications 3-46
3.2.11.1	User Subroutine Data Modifications 3-49
3.2.12.	Defining and Executing a GIPSY Process 3-49
3.2.13.	Executing GIPSY in the Batch Environment 3-54
3.2.13.1.	GIPSY Batch JCL 3-54
3.2.13.2.	GIPSY Batch Limitations 3-55
3.2.14.	Parameterized Input 3-56
3.3	Statistical Reports 3-56
3.3.1	Building Statistical Reports 3-59
3.3.1.1	Row and Column Definitions 3-59
3.3.1.1.1	USE Process 3-60
3.3.1.1.2	SELECT Process 3-62
3.3.1.1.3	RANGE Process 3-63
3.3.1.1.4	EXPLICIT Process 3-64
3.3.1.1.5	Specifying Calculated Data Values 3-65
3.3.1.2	Category and Section Definitions 3-66
3.3.1.3	Defining Multiple Reports 3-66
3.3.2	Displaying Statistical Reports 3-67
3.3.2.1	Tabular Report 3-69
3.3.2.2	Bar Graphs 3-81
3.3.2.3	Histograms 3-91

Section		Page
3.3.2.4	Point and Line Graphs	3-91
3.3.2.5	The Curve Graph	3-99
3.3.2.6	The Step Graph	3-103
3.3.2.7	Gantt Chart	3-107
3.3.2.8	Pie Charts	3-111
3.3.3	Modifying Statistical Reports	3-116
3.3.3.1	Matrix Modification	3-116
3.3.3.1.1	Limiting Rows, Columns, Sections and Categories	3-116
3.3.3.1.2	Report Totals	3-120
3.3.3.1.3	Vector Sequencing	3-122
3.3.3.1.4	Accumulating Vector Values	3-122
3.3.3.1.5	Limiting Number of Decimal Places	3-123
3.3.3.2	Establishing Default Graph Parameters	3-123
3.3.3.2.1	Fill Specifications	3-124
3.3.3.2.2	Shading Specifications	3-124
3.3.3.2.3	Proportion Specifications	3-125
3.3.3.3	Preserving User Supplied Defaults	3-125
3.3.3.3.1	Control of Fill vs. Shading	3-125
3.3.3.3.2	Control of Axis Parameters	3-126
3.3.3.3.3	Establishing Axis Parameters	3-126
3.3.3.3.4	Adding Tic Marks	3-127
3.3.3.3.5	Adding Grid Lines	3-127
3.3.3.3.6	Preserving Axis Parameters	3-130
3.3.4	Enhancing Statistical Reports	3-132
3.3.4.1	Adding Symbol	3-132
3.3.4.2	Adding Textual Information	3-133
3.3.4.3	Graph Refresh Capability	3-135
3.3.4.4	Turning the Graph Off	3-135
3.3.5	Saving Statistical Reports	3-141
3.3.6	Accessing Saved Reports	3-141
3.3.7	Building a New Report	3-141
3.3.7.1	ASSIGN Function	3-143
3.3.7.2	DELETE Function	3-146
3.3.7.3	RENAME Function	3-147
3.3.7.4	SUBSET Function	3-148
3.3.7.5	CHANGE Function	3-149
3.3.7.6	DEFINE Function	3-153
3.3.7.7	ADD Function	3-154
3.3.7.8	INPUT Function	3-154
3.3.7.9	REVIEW Statement	3-157
3.4	Geographic Displays	3-157
3.4.1	Geographic Display Definition Statements	3-158
3.4.1.1	Map Definition	3-158
3.4.1.1.1	The FILE Clause	3-160
3.4.1.1.2	The WITH Clause	3-160
3.4.1.1.3	The WINDOW Clause	3-166
3.4.1.1.4	The COLOR Clause	3-168
3.4.1.1.5	The PROJECTION Clause	3-168

Section	Page
3.4.1.2	Area Names 3-168
3.4.1.3	Location Names 3-171
3.4.1.4	Geographic Display Control 3-171
3.4.1.5	Geographic Grids 3-173
3.4.2	Data Base Geographic Displays 3-174
3.4.2.1	Symbol Plots 3-174
3.4.2.1.1	User-Defined Graphic Characters 3-179
3.4.2.2	Building a Track Plot 3-184
3.4.2.3	Synthesizing the Geographic Data 3-188
3.4.3	Maps and Data Displays 3-188
3.4.3.1	Viewing the Display 3-190
3.4.3.2	Interactive Display Building and Modification 3-190
3.4.3.2.1	Adding Tracks 3-190
3.4.3.2.2	Adding Symbols 3-193
3.4.3.2.3	Adding Circles 3-193
3.4.3.2.4	Adding Geodetic Information 3-194
3.4.3.2.5	Changing the Viewed Area 3-195
3.4.3.2.6	Limiting the Data Displayed 3-197
3.4.3.2.7	Listing the Display Data 3-198
3.4.3.2.8	Adding Textual Information 3-198
3.4.4	Secondary Data Retrieval 3-199
3.5	GIPSY Language Mode Transition 3-200
3.6	Picture Processing 3-200
3.6.1	Saving Pictures 3-200
3.6.2	Recalling Pictures 3-201
4.	QUERY PROCEDURES 4-1
4.1	File Query 4-1
4.1.1	Data Retrieval 4-1
4.1.2	Browsing 4-1
4.1.3	Subsetting the Database 4-3
4.1.4	Printing to a File 4-3
4.2	Data Sorting 4-4
4.3	Data Modifications 4-4
4.3.1	Field Level Data Modification 4-5
4.3.2	Record Level Data Modification 4-9
5.	USER TERMINAL PROCESSING PROCEDURES 5-1
5.1.	WYS/CUC Early Product Workstation and Z-248 PC/AT GIPSYmate Interface 5-1
5.2	MAGIC/GIPSY Interface 5-1
6.	GIPSY's GENERALIZED DATA REPORTS 6-1
6.1	Initiating the Generalized Data Reports Module 6-1
6.2	Data Manipulation 6-1
6.2.1	Field Manipulation 6-1
6.2.1.1	Add Type 6-2
6.2.1.2	Command to Add Field Definition 6-2

Section		Page
6.2.2.1	Database Sort	6-2
6.2.2.1.1	SORT Command	6-2
6.2.2.1.2	RESORT Command	6-3
6.2.2.2	Data Record Qualification	6-3
6.2.3	QDF Manipulation	6-3
6.3	Report Building	6-4
6.3.1	Report Definition	6-4
6.3.1.1	Parts Definition	6-4
6.3.1.1.1	Header Table	6-4
6.3.1.1.2	Body Table	6-4
6.3.1.1.3	Trailer Table	6-10
6.3.1.1.4	Output Element Statements	6-10
6.3.1.1.4.1	The LINE Statement	6-10
6.3.1.1.4.2	The SPACE Statement	6-19
6.3.1.1.4.3	The EJECT Statement	6-20
6.3.1.1.5	The Output Table Subroutines	6-21
6.3.1.1.5.1	\$BODY-LINE	6-21
6.3.1.1.5.2	\$BODY-PART	6-21
6.3.1.1.5.3	\$LINES-LEFT	6-22
6.3.1.1.5.4	\$PAGE-NUMBER	6-22
6.3.1.1.5.5	\$PRINT-DATE	6-22
6.3.1.1.5.6	\$PRINT-TIME	6-23
6.3.1.1.5.7	\$TOTAL-PAGES	6-23
6.3.1.2	Group Definition	6-23
6.3.2	Report Preparation	6-23
6.4	Report Viewing	6-24
6.4.1	Interactive Book Review	6-24
6.4.2	Non-Interactive Book Review	6-25
6.4.3	Book Review Display Commands	6-30
6.5	Directing Book Output to Printer	6-37

APPENDIXES

A.	GIPSY EXECUTION SEQUENCE	A-1
B.	SYSTEM SUPPLIED SUBROUTINES	B-1
C.	FIELD TABLES	C-1
D.	BUILD NEW REPORT EXAMPLE	D-1
E.	AREA NAMES	E-1
F.	LOCATION NAMES	F-1
G.	WIS WORKSTATION (WWS) GIPSY TERMINAL OPERATIONS	G-1
H.	INDEX	H-1
	DISTRIBUTION	I-1
	STANDARD FORM 298	J-1

ILLUSTRATIONS

Figure		Page
1-1	List of Airfield Information	1-3
1-2	Geographic Display of Airfield Information	1-4
2-1	GIPSY Configuration at JDSSC	2-4
2-2	System Organization	2-8
2-3	Error Correction Sequence	2-10
2-4	Structural Organization of GIPSY Language	2-11
2-5	Tabular Report	2-13
2-6	Bar Graph	2-14
2-7	Histogram	2-15
2-8	Point Graph	2-16
2-9	Line Graph With Single Line	2-17
2-10	Line Graph With Multiple Lines	2-18
2-11	Pie Chart	2-19
2-12	BNR Report Modification	2-20
2-13	Modified Report	2-21
2-14	Last Page of Modified Report	2-22
2-15	GIPSY World Map With Grid Lines	2-24
2-16	Symbol Plot	2-25
2-17	Track/Symbol Plot	2-26
2-18	Zoom in on Track and Symbol Plot	2-27
2-19	Result of Zoom	2-28
2-20	Window for Geographic List	2-29
2-21	Geographic List	2-30
2-22	Sample Printed Report	2-31
2-23	Sample Formatted Report	2-32
3-1	VIP 7705 and Tektronix 4014 Log-on Procedures	3-2
3-2	Sample of PCS Statement	3-13
3-3	Index/File Relationship	3-31
3-4	Graph of $y=2x+1$	3-58
3-5	Example of BREAK Command	3-68

Figure		Page
3-6	GIPSY Page Numbering	3-70
3-7	Bar Graph Using Specific Column	3-83
3-8	Bar Graph Using Specific Row	3-84
3-9	User Shading on Group Bar Graph	3-87
3-10	Stacked Bar Graph With Shading Densities	3-89
3-11	Stacking and Using Bar Graphs	3-90
3-12	Paged Bar Graph	3-92
3-13	Sample Histogram	3-96
3-14	Line Graph With Using Phrase	3-100
3-15	Tabular Report for Curve Examples	3-102
3-16	Sample Curve Display	3-104
3-17	Sample Step Graph	3-105
3-18	Step Graph Data	3-108
3-19	Step Graph With SHOW Option	3-109
3-20	Basic Gantt Chart	3-112
3-21	Sample Pie Chart	3-114
3-22	Pie Chart With USING and FOR Headers	3-115
3-23	Pie Chart With LABEL Options	3-117
3-24	Appended Totals	3-121
3-25	Result of SCALE Command	3-128
3-26	Result of TIC Command	3-129
3-27	Result of GRID Command	3-131
3-28	Initial Graphic Display	3-136
3-29	Positioning TXT1 With Graphic Cursor	3-137
3-30	Graph With TXT1	3-138
3-31	Positioning TXT2 With Graphic Cursor	3-139
3-32	Refreshed Graph With TXT1 and TXT2	3-140
3-33	Tabular Report Input to Subset	3-150
3-34	New Report After Subset	3-151
3-35	USA Map File	3-161
3-36	Software Zoom on Great Lakes Area From USA Map File	3-162
3-37	Software Zoom on Chesapeake Bay	3-163
3-38	Software Zoom on Virginia and Maryland From WORLD2 Map File	3-169
3-39	Geographic Circles	3-178
3-40	Standard System Plot	3-180
3-41	Track With Distance Option	3-187
3-42	Protected Symbol and Track Plot on WORLD2 Map File	3-189
3-43	Keyboard Supplied Tracks, Circle, and Symbol	3-192
6-1	Sample PCS for BOOK Named BKAIRFIELD	6-5
6-2	Display of GROUP Named INDEX	6-26
6-3	Display of GROUP Named RUNWAY	6-31
C-1	PCS With GIPSY Subroutines	C-28
C-2	Output From BOOK-PLACE, PART-WINDOW	C-30
C-3	Output From BOOK-PLACE, PART-LOCATION	C-31
G-1	GIPSY Undefined Terminal Prompt	G-5

TABLES

Number	Page
3-1	WORLD2 Political and Topographic Map Details 3-164

ABSTRACT

The Graphic Information Presentation System (GIPSY) is a general purpose graphics and information display capability. It combines the tools of data retrieval, information processing, formatted reports, tabular, graphic, and geographic display into a single integrated on-line interactive system. It is a file and data independent system that is driven by a high level user oriented language. The graphic display capabilities were implemented using a device independent approach which allows the single integrated system to support multiple dissimilar devices. GIPSY automatically reconfigures itself to the capabilities and unique requirements of the terminal to which the user is logged onto.

The graphic display capabilities were the primary basis for the initiation of this system. However, GIPSY very effectively serves as an information handling system to connect the user's data base to a large set of on-line interactive query and display capabilities.

This document provides a mid-level tutorial description of the capabilities of GIPSY. GIPSY is the WWMCCS Standard Operating System software for computer graphics as approved by System Development Notification (SDN) K7804. This document corresponds to GIPSY Release 5.2 which operates with Honeywell's General Comprehensive Operating System (GCOS) 8.

This manual supersedes Computer System Manual (CSM) Users Manual (UM) 259-85 dated 15 August 1985.

SECTION 1. GENERAL

1.1 Purpose of the Users Manual

The objective of this Users Manual is to provide non-ADP personnel with the information necessary to effectively use the Graphic Information Presentation System (GIPSY). This includes familiarizing the reader with the concepts behind interactive on-line computer graphics. Advanced concepts involving interface programs in the use of GIPSY are not discussed in this manual.

This Users Manual is organized such that each GIPSY user is required to learn only that part of GIPSY that is related to his/her applications. Section 1 defines the project and provides background information necessary for understanding this document. Section 2 provides an overview of the system. Section 3 describes the language used by GIPSY and its associated outputs. These three sections should be read by each GIPSY user. Sections 4 and 5 are aimed toward the applications programmer who wants to use GIPSY in conjunction with existing information and query systems. Section 6 focuses in on the specifics of interfacing with the WWMCCS Information System (WIS) Early Products Workstation and the Zenith 248 PC/AT. Section 7 explains GIPSY's Generalized Data Reports module.

1.1.1 Project Orientation

GIPSY is an outgrowth of the effort to satisfy the graphic needs of the National Military Command System (NMCS). Capability requirements were extracted from Technical Support Requirement (TSR) documents, requirement studies, and related documents identified under subsection 1.2. Although GIPSY was originally oriented toward the NMCS user, it is equally applicable to other WWMCCS sites. An overriding concern of this project was to produce an information presentation and graphic capability that is independent of any particular application, yet able to effectively interface with that application as it currently exists.

1.1.1.1 Introduction to Graphics. It has been said that a picture is worth a thousand words. Few people would argue with that statement. We have only to look at our newspapers, magazines, and TV sets to appreciate the importance of pictorial presentations of information. A photograph records information as it exists in a given instant in time; unfortunately, storage, recall, and update of that information is nearly impossible using photographic methods. To solve this problem we strip the important pieces of information from reality and simulations. Then we store the information in digital form in computer memory. Now the problem of storage, recall, and update is solved, but we have a presentation problem. Presenting the information in a printed or textual form solves part of the problem but by no means all of it. Valuable concepts and interrelationships are lost or are not detected because the information is presented as a list of data rather than a concise image of the relevant set of data.

Compare a printout of geographic coordinates and unit identification

(figure 1-1) with a map having each unit clearly plotted at its respective location (figure 1-2). The contrast is striking indeed; but it clearly relates the graphic presentation method to the normal printed report method. We could go on discussing the merits of graphic presentation over textual methods; however, there are a number of documents (see subsection 1.2) which more eloquently address this subject than we can within the scope of this document. We invite you to make comparisons of textual methods and the graphics produced in the illustrations in this manual.

1.1.1.2 Purpose of Project. The primary purpose of GIPSY is to provide a basic, user analyst-oriented, graphics capability for displaying data extracted from a variety of data sources in both graphic and alphanumeric form. A secondary purpose is to establish a frame of reference upon which a comprehensive graphics capability can be built based upon requirements defined and refined by the user community after exposure to the capabilities. The basic set of capabilities from known or documented WWMCCS and NMCS requirements was used as a baseline capability. The framework is established by which requirements not already addressed can be addressed in future releases of GIPSY.

1.1.1.3 Scope of Project. The scope of GIPSY is limited to the purposes defined above. GIPSY is not a vehicle for pure computer graphics. It is not a product for editing and creating slides from sketches or drawings. More properly, GIPSY is a data-based system application of computer graphics. The primary thrust of on-line, interactive computer graphics, (or simply graphics, for short) within the command and control community is the presentation of data in the form of tabular information summaries, management graphs, statistical graphs, and geographic displays. GIPSY provides a set of operational tools that allow the applications analyst to build a user-friendly interface between the staff-level user and his/her data system. This methodology effectively allows the application of GIPSY to be tailored to a specific set of information or user analyst.

1.1.1.4 Audience. GIPSY is targeted at the audience of users at least one step removed from the programmer. GIPSY is primarily oriented toward the decision maker who knows his/her data but not the intricacies of data processing. Unlike most graphic systems, GIPSY is not a subroutine library with which the user writes "simple programs" to generate graphics. There is not a long list of reserved words or codes the user has to memorize. In fact, there are no reserved words. GIPSY commands are English-like sentences whose words assume meanings dependent upon the context in which they are used. The GIPSY user does not have to learn the entire GIPSY language, only that subset defining his/her functions.

1.1.1.5 Essential Considerations. The user feedback mechanism is an extremely important feature in the development and maintenance of GIPSY. It is the vehicle for influencing the development process. The GIPSY management staff does not presume to anticipate all users' requirements or to dictate what the users must accept. The modular construction of GIPSY makes it easy to adapt the system to unique applications while maintaining compatibility

COORDINATE STATUS LENGTH SURFACE CONDITION

NAME

CECIL FIELD NAS	301258N0815230W	MILITARY	12500	ASPHALT	GOOD
ORLANDO INTERNATIONAL	282554N0811929W	CIVIL	12004	CONCRETE	GOOD
EGLIN AFB	302912N0863134W	MILITARY	12000	ASPHALT	GOOD
HOMESTEAD AFB	252917N0802302W	MILITARY	11200	CONCRETE	GOOD
MACDILL AFB	275057N0823118W	MILITARY	11420	ASPHALT	GOOD
MIAMI INTERNATIONAL	254733N0801710W	CIVIL	13002	ASPHALT	GOOD

Figure 1-1. List of Airfield Information

UNCLASSIFIED

FLORIDA AIRFIELDS (RUNWAY LENGTHS GREATER THAN 11000 FEET)

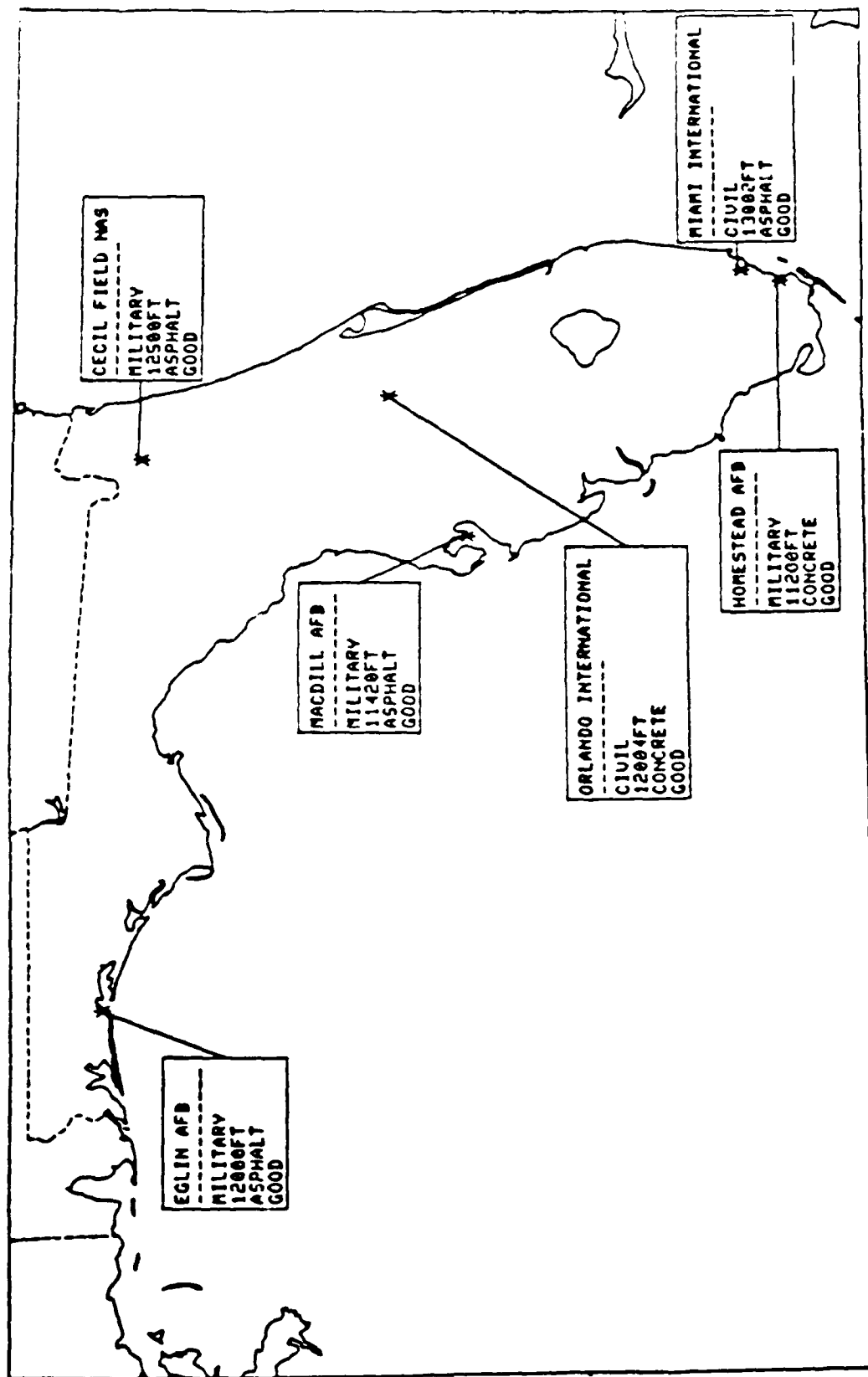


Figure 1-2. Geographic Display of Airfield Information

with subsequent releases of the system containing more powerful capabilities.

1.2 Project References

- a. Terra Plot System (TPLOT)
Computer System Manual
CSM UM 161-82, 30 July 1982
- b. Data Presentation System (DPS)
Computer System Manual
Number CSM UM 39B-68, 20 August 1971
- c. Technical Report Number TR 116-77, Interactive Computer Graphics, Paul R. Davis, Major, USAF, and John R. Scott, Captain, USA, 20 June 1977
- d. J3M-461-77, WWMCCS Graphics Requirements, 24 February 1977
- e. NMCS TSR, Technical Support Requirements for FY 79-83
Common and Indirect Support, Joint Chiefs of Staff
21 February 1977.
- f. SDN K7613, CCTC Prototype Graphics System, 5 May 1976
- g. SDN (NO. 7770612) Tektronix Graphic Terminal and Copier for SAGA, 26 April 1976
- h. SDNs K7515, K7516, NMCS Information and Display System, 19 June 1975
- i. CCTC Memo, CCTC Graphics Review, 15 April 1977
- j. DCA Contract No. PR C421-76-37, Research, "Toward Effective Resource Sharing: Networking Research in Front Ending and Intelligent Terminal," 17 September 1976
- k. W. T. Gorse and S. S. Poh, "Final Report on Requirements Analysis for WWMCCS Displays and Source Data Automation," MITRE Corp., MTR-7324, DCA/WAD F19628-76-C-0001, August 1976
- l. T. L. Elliott and S. S. Poh, "Graphics Development Plan for WWMCCS," MITRE Corp., MTR-5273, DCA/WAD F19628-76-C-0001, September 1976
- m. CCTC TM 146-77, CCTC ADP Management Plan, Volume II, 1 March 1977
- n. NMCSSC Memo, Graphics Requirements for NMCSSC, 1 October 1975
- o. BIS7705-TM Emulator Manual, Version 4.0 Micro-Integration, Inc., ML-1500-02-PI, 1986.

1.3 Terms and Abbreviations

Abscissa - The set of x data values of a graph. Generally the vector (row or column) headers are used as the abscissa; the abscissa cannot be specified in GIPSY except on the curve description. If the ordinate is a row vector the abscissa will be the column headers. The converse is also true.

ASCII - American Standard Code for Information Interchange. This is the 4-characters per word TSS operational mode.

Baud - Information transfer rate normally meaning bits transferred per second.

BCD - Binary Coded Decimal. This is the 6-bits per character code that is normal to DPS-8/DPS 8000 batch mode operation.

Cat/File String - The string of characters that is used to uniquely identify a data file on the DPS-8/DPS 8000. The cat/file string is sometimes referred to as catalog file descriptor (CFD).

Catalog File Descriptor - See Cat/File String.

COMM - Abbreviation for communication; See line mode.

Communications Mode - See line mode.

CRT - Cathode Ray Tube. This term is used to refer to a nonprinting (or hardcopy) display terminal.

DAFC - The Directive Action and File Control (DAFC) file is a GIPSY generated temporary file which contains the status of all GIPSY parameters and user input specifications. The DAFC is the central file in the system. All intermodule communication occurs through the DAFC; The GIPSY command word which declares the intent to specify that a previously saved DAFC file is to be restored with execution resuming at the point in which the DAFC was saved.

DISA - Defense Information Systems Agency, formerly Defense Communications Agency (DCA).

DSSO - Defense Systems Support Organization, formerly Joint Data Systems Support Center (JDSSC).

Element - A number in the body of the tabular report (matrix).

FDT - A File Descriptor Table enables the user to describe his data file to GIPSY. The FDT may be specified in-line or recalled from a previously saved file.

FST - The File Structure Table that is created by the GET IDSII STRUCTURE. It displays the record type identifiers and the relationships among the different record types of an I-D-3/II integrated file only.

GCOS - General Comprehensive Operating System. The WWMCCS operating system for the DPS-8/DPS 8000.

GDS - Graphic Data Set (GDS) is a file containing core image representation of one or more tabular reports. The GDS is normally created by the matrix generation module using the QDF as input; The GIPSY command word used to declare an alternate file for the GDS as defined above.

THIS PAGE INTENTIONALLY LEFT BLANK

GFRC - General File and Record Control. One of the types of file systems used on the DPS-8/DPS 8000.

H6000 - The generic name for the family of 6000 series computers produced by Honeywell Information Systems, Inc. It is now referred to as DPS-8/DPS 8000.

I-D-S/II - Integrated Data Store/II. May be either an indexed or integrated file format on the DPS-8/DPS 8000.

Indexed File - A UFAS indexed file format uses key fields that are defined during the creation of the data file to expedite data retrieval.

Integrated File - An I-D-S/II integrated file format may consist of many different record types organized in a hierarchy that consists of one or more owner record types that may have subordinate record types, which may in turn have subordinate record types.

Interactive - Having the property of immediately responding to inputs in a conversational manner. A process is said to be interactive only if each complete information unit supplied is responded to and acted upon prior to proceeding to the next step. The interactions have the ability to dynamically alter the course of the conversation and objective thereof.

Interrupt - An action which gets the immediate attention of current program process. Normally a different course of action is effected.

ISP - Indexed Sequential Processor. A GFRC file organization that allows access to records in both sequential order and random access through the use of a unique key value.

JDSSC - Joint Data Systems Support Center. Now referred to as Defense Systems Support Organization (DSSO).

Line Mode - A setting on a terminal in which the terminal is able to send data to and receive data from the computer. This is usually set by a contact switch or toggle switch on the terminal labeled "line" or "COMM".

Logic Table - The Logic Table is a passive tool in GIPSY which allows the user to assign any number of conditional expression to a character name of 1 to 12 characters; Names defined in the logic table may be used in lieu of a conditional expression in any statement which allows for a conditional expression, including the logic table itself.

Null File - A named area for which space has been allocated, but which contains no information, not even an end of file.

Math Table - The Math Table does for arithmetic expression what the logic table does for conditional expressions.

Mode - The mode is the name assigned to the dimensions of a tabular report. The four modes are COLUMN, ROW, CATEGORY and SECTION.

On-line - Having direct and immediate connection to the computer.

Ordinate - The set of y data values of a graph. In all GIPSY graphs the vector defining ordinates must be explicitly stated.

Ordinate Descriptor - The vector header that defines the vector containing y values.

Partial Field Notation - A technique for selecting a part of a previously defined field. It is a pair of numbers specified in parenthesis following a field name; the first number identifies the first character position within

THIS PAGE INTENTIONALLY LEFT BLANK

the field of data to be used and the second number identifies the character position within the field of the last character to be used.

PCS - The Process Control Statement (PCS) is a user file designated as containing GIPSY source language statements; The GIPSY command word which declares the intent to specify a catalog-file string of a file containing GIPSY source language statements.

Permanent File - A cataloged data set whose name and location are known by the system and may be accessed by referencing that name.

Prompt Character - A character that is displayed to indicate the on-line interactive input devices is ready to accept input. In GIPSY the character is a ">", and in TSS it is "*".

Protocol - The mechanism through which one computer or device communicates with another. This is information transfer above and beyond that which is displayed on the screen.

QDF - The Qualified Data File (QDF) is a data file which contains the minimum subset of the user's data file needed to produce the required statistical or graphical display (see appendix E for full discussion); The GIPSY Command Word which declares the intent to specify an alternate file for the QDF defined above.

QDT - The Qualified Descriptor Table (QDT) is an internal GIPSY data structure which describes the QDF.

Random File Access - A method of file access that involves the retrieval of a specific file record by referring to a unique record identifier.

Relative - A (UFAS) file format that has a unique number associated with each record that can be used to identify a specific record to be retrieved.

Sequential File Access - A method of file access that involves the retrieval of data records from a file in the logical order in which they are stored in that file.

Tabular Report - An organization of data into rows and columns with a numeric value at the intersection of each row and column. The tabular report also contains classification and title information.

Temporary File - A data set which exists only during the duration of a terminal session.

TSS - Time Sharing System. This is the Standard DPS-8/DPS 8000 System for on-line interactive processing.

UFAS - Unified File Access System. A DPS-8/DPS 8000 file management system.

Vector - A group of elements comprising a single ROW or COLUMN.

Vector Header - The name associated with a single COLUMN, ROW, CATEGORY or SECTION. Usually referred to with the modes (e.g., ROW, Headers, SECTION Headers).

WWMCCS - Worldwide Military Command and Control System.

WWDMS - Worldwide Military Command and Control System Data Management System.

THIS PAGE INTENTIONALLY LEFT BLANK

1.4 Security and Privacy

GIPSY has no classified data files and thus has no special provision for file security and privacy. Marking of outputs produced by GIPSY is controlled by commands provided by GIPSY. The standard security caveat features cannot apply to graphics since they are based upon counting line feed characters. Further details are provided in the discussion of the GIPSY Classification Statement, subsection 3.2.3.

GIPSY does create temporary files which contain fragments of user data, and it can create permanent files in user file space when so requested. This should be considered when choosing your logon classification since GIPSY uses the standard Time Sharing System (TSS) facilities for file creation and access.

THIS PAGE INTENTIONALLY LEFT BLANK

SECTION 2. SYSTEM SUMMARY

This section describes the application, operation, configuration, organization, performance, inputs and outputs of GIPSY.

2.1 System Overview

The Graphic Information Presentation System (GIPSY) is designed for WWMCCS users with a need for timely displays of simple or complex information in the form of tabular reports, graphs, and geographic displays. GIPSY's problem-solving capabilities provide for on-line, interactive terminal use on the Honeywell Information System (HIS) 6080 computer system. Highlights of GIPSY include the following:

- a. Language Simplicity - GIPSY's user language has an English-like vocabulary and grammar. Processing is controlled by powerful context sensitive language statements such as -- RETRIEVE a set of data -- BUILD a tabular report -- or ZOOM to a larger or smaller map scale.
- b. Data Retrieval - Statements are provided to selectively retrieve a logical subset of the user's data file for synthesis into a tabular, graphic or geographic display. A secondary retrieval allows retrieval from the retrieval data.
- c. File Independence - The layout of the data in the user's data record is described to GIPSY once in a file descriptor table in which each field is assigned a name. From that point onward, the user references his/her data by the names assigned to the data fields.
- d. Tabular Displays - GIPSY produces reports in the form of rows and columns of numerical information, complete with row and column headings, titles, security classification, and page numbers. The user can determine the numerical information to be included in the columns of the table in a number of ways. The user may choose to simply cross-tabulate two or more data fields, or to build a complex report whose entries are conditionally calculated values.
- e. Data Browsing - The secondary retrieval combined with GIPSY's alphanumeric display capability provides an effective tool for browsing through selected portions of the data retrieved.
- f. Report Modifications - The tabular report is structured as a matrix which can be transformed in either row or column mode or in both modes using arithmetic operations (+, *, /, -) and predefined functions. Vectors (rows or columns) can be added, deleted, or modified. Individual values may be changed.
- g. File Subsetting - A subsetting capability is provided to allow the user to select and save a subset of his data base after applying retrieval and data modification procedures.

- h. Geographic Maps - A map of any portion of the world (currently using Mercator and Lambert projections) can be displayed and automatically scaled to fit the display area. Location and track plots can be superimposed on these projections.
- i. Geographic Location Plot - GIPSY will plot a symbol at the location designated by the contents of a coordinate field extracted from data records. This is very effective for displaying the current location of a unit and distinguishing each unit by different symbols based on data content or selection criteria.
- j. Geographic Track Plot - Given a coordinate field and a track identifier, GIPSY will draw a geographic track by connecting the points in a tracking sequence.
- k. Geographic Display Function - Geographic display functions, such as zooming and listing contents of data points on the display, add to the flexibility of GIPSY's powerful geographic capabilities.
- l. Device Independence - Device characteristics that are unique are isolated from GIPSY and are resolved at execution time. This capability maximizes the independence of GIPSY from any particular vendor's device. GIPSY can be used at any H6000 site using the standard TSS. Installation is simple, and ease of maintenance is assured by GIPSY's modular construction.
- m. In-line Data Modification - Data in operational data bases often requires supplemental data or restructuring of existing data in order to maximize the utility of the data and its presentation. GIPSY provides tools to dynamically restructure the data after it is read from the user's data file but prior to any GIPSY processing of that data. Consequently, discovered data error can be corrected, supplemental data added, etc.
- n. Save and Restore Execution - At any time during the execution of GIPSY the user may "freeze" the GIPSY execution and save it on a file for a subsequent restoration of the process at the point it was interrupted.
- o. Display Save/Recall - Completed graphic displays may be saved and subsequently recalled and redisplayed. Graphic display may even be created and saved in the batch mode and recalled as pictures in an on-line mode.
- p. Clear Text Messages - GIPSY never provides the user with coded messages or error numbers. All messages are presented in concise English sentences. Messages are available not only in event of an error in a GIPSY input statement but the system contains supporting messages which are always available to let the user know what input options are available.

2.2 System Operation

GIPSY operates in an interactive, time-sharing mode. It accepts input, processing, and output specifications in the form of language statements entered via a terminal keyboard, from a previously saved file, or from a combination of both. The internal operations of the system do not require the user to enter job control statements or to be otherwise directly involved in processing functions as is the case in batch, card-oriented systems and applications. Batch execution of GIPSY is available for applications requiring large volumes of output.

From the perspective of the GIPSY user, system operation consists of keying in language statements and responding to GIPSY messages. As each statement is entered, it is immediately edited, and the user is notified if an error was made. Errors are identified and explained and the user is instructed in the recovery from the error condition.

GIPSY requires no special permissions, priority, privity, or special consideration from the operation environment. In fact, GIPSY's operation is sufficiently simple that it can be initiated and executed by a user's system or FORTRAN program.

2.3 System Configuration

GIPSY operates on Honeywell 6000 series computers as a subsystem under the Time-Sharing System (TSS). It executes within 12K to 32K words of TSS memory. Each user of GIPSY gets a separate copy of the system, but by virtue of operating in the TSS region of memory each user may be swapped out when either that user is idle or a higher priority request comes in. Since an interactive system is idle most of the time the cost of having each user get his own copy of GIPSY is minimized. Disk storage requirements are minimal. No preassigned output files are required since GIPSY acquires temporary file space for work and data storage.

Although GIPSY was developed using a Tektronix 4014-1 graphic display unit, the software was written with minimal dependence upon any particular terminal hardware. The non-graphic output can be produced on any American Standard Code for Information Interchange (ASCII) terminal. Once the device is described to GIPSY (via a device definition table) output is automatically formatted for that device. A device definition table exists for the Tektronix 4014, 4027, 4107, VIP 7705 and VIP 786, WIS Early Products Workstation (WIS/CUC), Zenith 248 AT/PC, WIS Workstation (MAGIC), and Batch.

The WIS/CUC emulates a Tektronix 4014. Users should logon as they would for any WIS/CUC-H6000 session and GIPSY will automatically switch the terminal to 4014 mode.

If a program-controllable hardcopy device is available, GIPSY can use it to automatically produce hard copies from a Cathode Ray Tube (CRT) type device. Figure 2-1 illustrates a configuration used in the JDSSC.

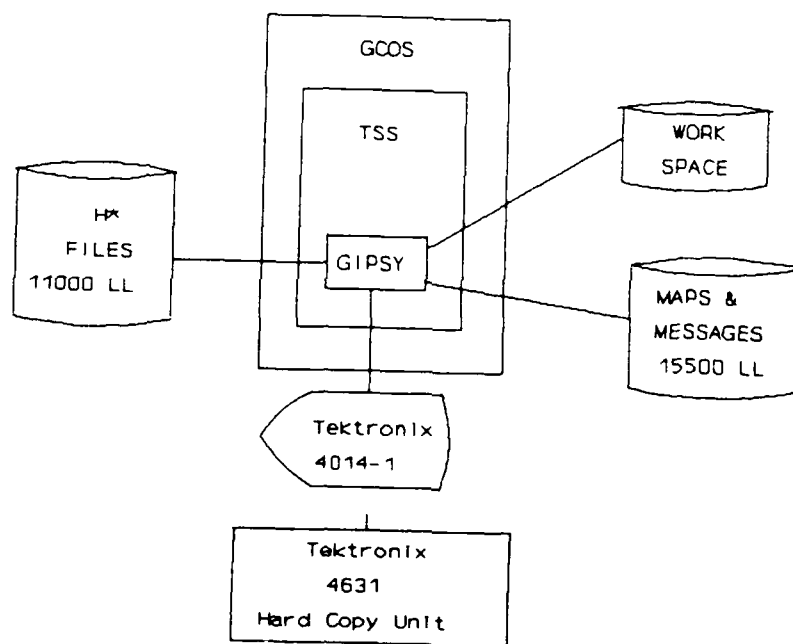


Figure 2-1. GIPSY Configuration at JDSSC

Minimum configuration is an H6000 with sufficient memory to execute a 32K word program, work and permanent file space, and alphanumeric input/output terminal. Optimal configuration includes a graphic input/output terminal, and a program-controlled hardcopy device. All functions of GIPSY except actual graphic output may be produced on a minimal configuration.

2.4 System Organization

The H6000 Time-Sharing System is organized as a series of independently executable object modules each of which may be executed by TSS or a subsystem under TSS. GIPSY parallels TSS in its organization in that major functions in GIPSY are independently-executable object modules. Each module can be automatically or explicitly executed by the GIPSY executive or by a user-written executive routine.

The user communicates with GIPSY by means of an alphanumeric graphic terminal for tabular, graphic, and geographic output. Any standard alphanumeric terminal may be used to create graphic data structures if only tabular output is desired. The user-input language is used to direct and control various capabilities of the system through an interactive process by which the user establishes the criteria for retrieval and organization of his data. From these input language statements GIPSY creates fixed data structures (called Process Control Tables) to control the various capabilities of the system.

2.5 System Performance

Once the data has been synthesized into a concise, graphic data structure, the system begins building the display immediately, and it is ready in seconds--depending on the speed of the graphics device and transmission speed. Input language specification to GIPSY is interactive; if an error is made, the system redisplayes the input together with a pointer to the segment of the statement in error. The user may simply type in a correction and the system will take care of the rest. This provides a highly effective method of specifying the graphic request including query and synthesis requirements.

Once graphic data structure is completed (i.e., the data is synthesized), the entire set of graphic outputs is available without any further data file processing.

2.6 Contingencies and Alternate Modes of Operation

Since GIPSY is a subsystem on the H6000, it has no alternate modes of operations of its own.

2.7 Database/Data Bank

Other than map data bases, supporting information lists, and message library, GIPSY has no data base of its own, and does not dictate data base formats for use in GIPSY.

GIPSY processes the following file types:

- a. General File and Record Control (GFRC) Sequential Files
- b. GFRC Indexed Sequential Processing (ISP) Files
- c. Unified File Access System (UFAS) Indexed Files
- d. Unified File Access System (UFAS) Relative Files
- e. Unified File Access System (UFAS) Sequential Files
- f. Integrated Data Store/II (I-D-S/II) Indexed Files
- g. Integrated Data Store/II (I-D-S/II) Integrated Files.

GIPSY does not automatically handle interrelationships among different record types. These interrelationships must be addressed by the user. GIPSY processes all records against all applicable GIPSY statements. The data file can be either batch-based Binary Coded Decimal (BCD) files or TSS-based ASCII data files. If the data base is ASCII, GIPSY will correctly interpret it as if it were BCD.

The data file is described to GIPSY via a structure called a File Descriptor Table (FDT). The FDT describes each field the user desires to reference and assigns a name to it. This name will be used to reference the defined data in the file.

A File Structure Table (FST), which is stored and accessed with the FDT, is created by GIPSY only for a UFAS I-D-S/II integrated file.

GIPSY is designed to allow you to use your data base as it currently exists. You may find it convenient to add data elements to your data file to take advantage of some of the more powerful automatic features of GIPSY.

2.8 General Description of Inputs, Processing, Outputs

This subsection of the Users Manual provides a semi-technical discussion of the mechanics of GIPSY. It is not necessary to understand the subsetting of files and information flow mechanics discussed here to utilize GIPSY effectively. This subsection also contains a brief summary of the input language and outputs of GIPSY.

2.8.1 Input and Processing Operations. The approach to producing graphic displays in GIPSY is to subdivide the tasks into individual units and then solve two or three smaller, well-defined tasks rather than one complex one. The steps involved are:

- a. Prepare or identify the data

- b. Describe the data file
- c. Define the subset of the data to be selected for processing
- d. Define the data syntheses process (report preparation instructions)
- e. Display the data.

THIS PAGE INTENTIONALLY LEFT BLANK

The first step involves identifying the data file to be used and becoming familiar with its contents. Inevitably, the outputs you define initially will be augmented by data displays not initially conceived, so it is important to be familiar with the data. The data is identified to GIPSY with a FILE statement which simply contains the catalog-file string of the desired data file.

In the next step, the data file is described to GIPSY in what is called a File Descriptor Table (FDT). Once the FDT is built, it is referenced by name from that point on. An FDT statement is used to tell GIPSY where the FDT is located. After GIPSY is told where the FDT is, the user can proceed to the subsequent steps of data selection and data syntheses.

The criteria for data selection are established by a RETRIEVE statement which has comparison operators (EQ, GT, GE, LT, LE, NE, BT, CHANGES, COMPLETE), a unary NOT operator, connectors (AND, OR), and parentheses (to control the hierarchy of comparisons, if needed). All data records which meet the retrieval criteria are saved in a temporary file called a Qualified Data File (QDF). The QDF and its associated Qualified Descriptor Table (QDT) are passed to the data synthesizers.

The QDF is a much smaller subset of the user's data file. It contains only data which is needed for subsequent use (i.e., it contains only referenced fields of qualified records). This is important because in many cases a large number of records will be needed but only a few fields. This technique ensures that a minimal amount of data need be handled by the data synthesizer and display modules. Further, it minimizes the time the user's master file is tied up, since GIPSY will release its access to the data file after the QDF is built. The user may spend an hour or so generating tabular, graphic, and geographic displays, and analyzing the retrieved data; during this time the user's data file is available for whatever functions need to be performed.

The QDF/QDT may be saved for subsequent GIPSY runs. This makes it possible to bypass selection of data from the master file if the data and retrieval have not changed.

Figure 2-2 puts the foregoing information into perspective. First log onto the H6000 Time-Sharing System (TSS) using standard log-on procedures (discussed under subsection 3.1); tell GIPSY's data selection and syntheses language processor (the SYNTAX module) where to find your file and its description. Then describe your tabular or graphic information collection procedures. The GIPSY commands input to perform these functions are translated and/or compiled and written to the Directive Action and File Control (DAFC) file to be accessed by the remaining modules in order to perform the requested functions.

The user's data file is attached and read by the Data Selection Module (DATSEL) and a QDF is produced. The QDT has already been created by the SYNTAX module by recording all data fields referenced in any statements. The QDF is created by copying the fields, defined in the QDT, from the data file

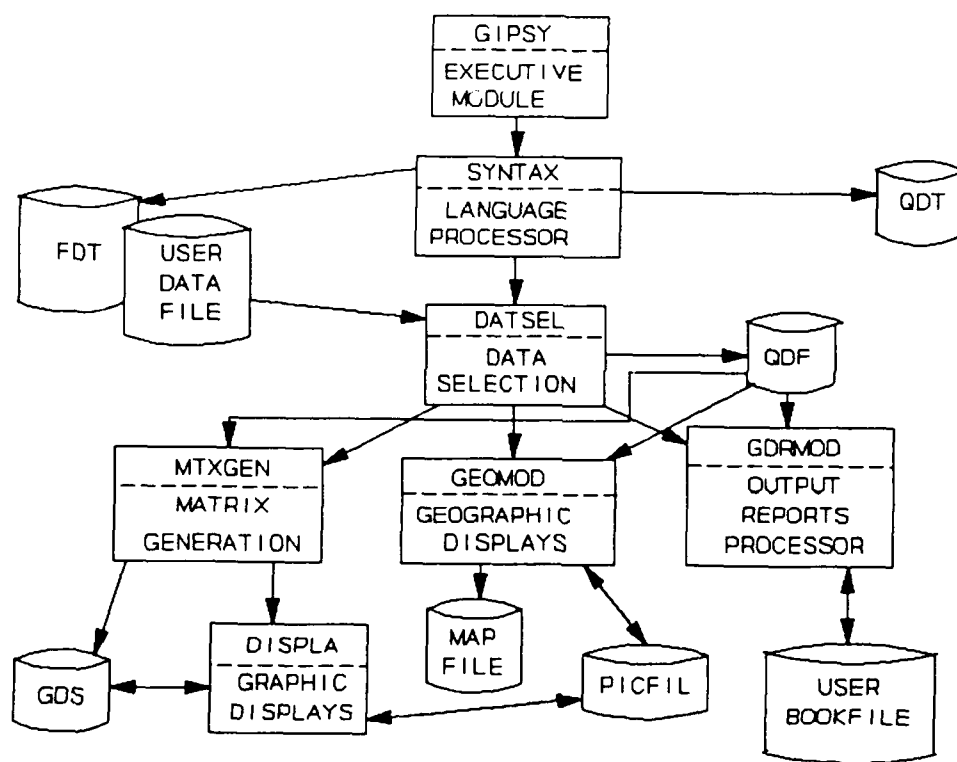


Figure 2-2. System Organization

to the QDF if the specified retrieval condition was met.

The QDF/QDT is then passed on to the data synthesis modules to be reduced to a concise graphic form called the Graphic Data Set (GDS). The GDS is the primary input to the Display Module (DISPLA). The QDF/QDT along with the map files are the primary inputs to the Geographic Display (GEOMOD) and the Generalized Data Reports (GDRMOD) Modules. The map files also serve as input in the Geographic Module.

In the current version of GIPSY, the GDS for the Geographic Display Module is not as well developed as the GDS for statistical portions. The geographic portion has a higher degree of dependence on the power of the graphic device; consequently, the Geographic Display Module relies more heavily on the QDF/QDT. The QDF/QDT are also the major input for the Generalized Data Reports Module.

2.8.2 Language Overview. The GIPSY Language is English-like in context and grammatical construction. The sentences begin with an imperative verb defining an action to be taken, or a declarative defining a data file or a mode of operation. There are no codes or reserved words; the meaning of each word in the GIPSY language is sensitive to the context in which it appears. Syntax validation is interactive. Each time you transmit a command to GIPSY it is immediately checked for syntactic validity; if an error is detected, the system will give an explanation as to the source of the error, print out the statement in error, and display pointers to the element of the statement in error. You may then simply retype the element in error; GIPSY will patch the correction into your statement and restart syntax validation at that point. The excerpt from an actual input and error sequence is shown in figure 2-3. Note that the correction was only for the portion of the statement flagged as incorrect.

GIPSY uses the symbol " > " as a prompt symbol to indicate that the keyboard console is open for user input.

The GIPSY language is a tree-structured language. The first word is a command word that identifies the branch of the tree; subsequent words further modify the command with additional branches and perhaps with some options (leaves) along the way. This definition of the language makes it natural, easy to remember, easy to follow, and easy to enhance without impacting other commands.

The GIPSY language falls into four categories: report descriptions syntax, graph and report displays, geographic displays, and formatted reports. These four categories also define the four modes of operation which are referred to respectively as SYNTAX, DISPLAY, GEO, and GDR. GIPSY is normally initiated in the SYNTAX mode. When the report is ready, you are placed in the DISPLAY, GEO, or GDR mode depending upon the type of report requested. There are commands to place you in either of these modes manually. Sections 3 and 6 of this manual discuss the language elements of these modes in detail. There is another feature that is not part of the language but is germane to

CLASS "VERY UNCLASSIFIED" (SIZE WRONG)
The indicated item is not a valid character size
valid sizes are JUMBO, LARGE, MEDIUM, SMALL. Enter either
the name or abbreviation for name (1st letter, etc).

CLASS "VERY UNCLASSIFIED" (SIZE WRONG)
 ^ ^

>JUMBO

>//LAST

CLASS "VERY UNCLASSIFIED" (SIZE JUMBO)

>

Figure 2-3. Error Correction Sequence

understanding the language. At any point in the language specification, the system knows what to expect as the next input. So, if you are ever uncertain as to what GIPSY is expecting as input - do what comes naturally - hit the carriage return key and GIPSY will tell you what it is expecting.

2.8.3 Outputs. This subsection defines and illustrates each of the basic outputs from GIPSY. Bear in mind that any number of permutations of each output may be possible.

- a. Tabular Report (figure 2-5) - This is the basic structure from which all the management graphs are produced. It is one of the elements from the Graphic Data Set (GDS). Associated with each tabular report are classification markings, title, row and column headers, and a matrix of values built from the user's input data. Except where noted, the x-axis data of the graphs are extracted from the row or column headers and y-axis data are the values in one of the vectors (row and column) of the matrix. All subsequently defined management graphs were generated from the tabular report in figure 2-5.
- b. Bar Graph (figure 2-6) - The bar graph produced by GIPSY may have a single bar for each observation along the x-axis or there may be several bars grouped side by side or, each stacked on top of the other with shading and corresponding legend to distinguish each. The system will pick a set of shadings to insure clearly distinguishable and neat shadings. Of course, the user may override the system with his/her own shading and density combinations.
- c. Histogram (figure 2-7) - The difference between a bar graph and a histogram lies primarily in the interpretation of the data shown on them. The observation shown on the x-axis of the histogram represents the midpoint of a set of values defined by the lateral limits of the bar. Visually, the only difference is the absence of space between the bars. Like the bar graph, the histogram may be stacked and shaded.
- d. Point Graph (figure 2-8) - At the y value of each observation on the x-axis, a user-specified symbol is plotted.
- e. Line Graph (figures 2-9 and 2-10) - The line graph is very simply a point graph with the succeeding points connected.
- f. Pie Chart (figure 2-11) - The pie chart shows the relationship of the individual vector elements to the total. The percentages are automatically calculated by GIPSY.
- g. Report Transformation/Modification (figures 2-12, 2-13, and 2-14) - Once a report has all the pieces of data collected into one place it is often necessary to perform arithmetic operations on the rows and/or columns of the old report to produce a new one. A "BUILD NEW REPORT" command exists for this purpose.

UNCLASSIFIED

1910 00 AUG 85
1 OF 1

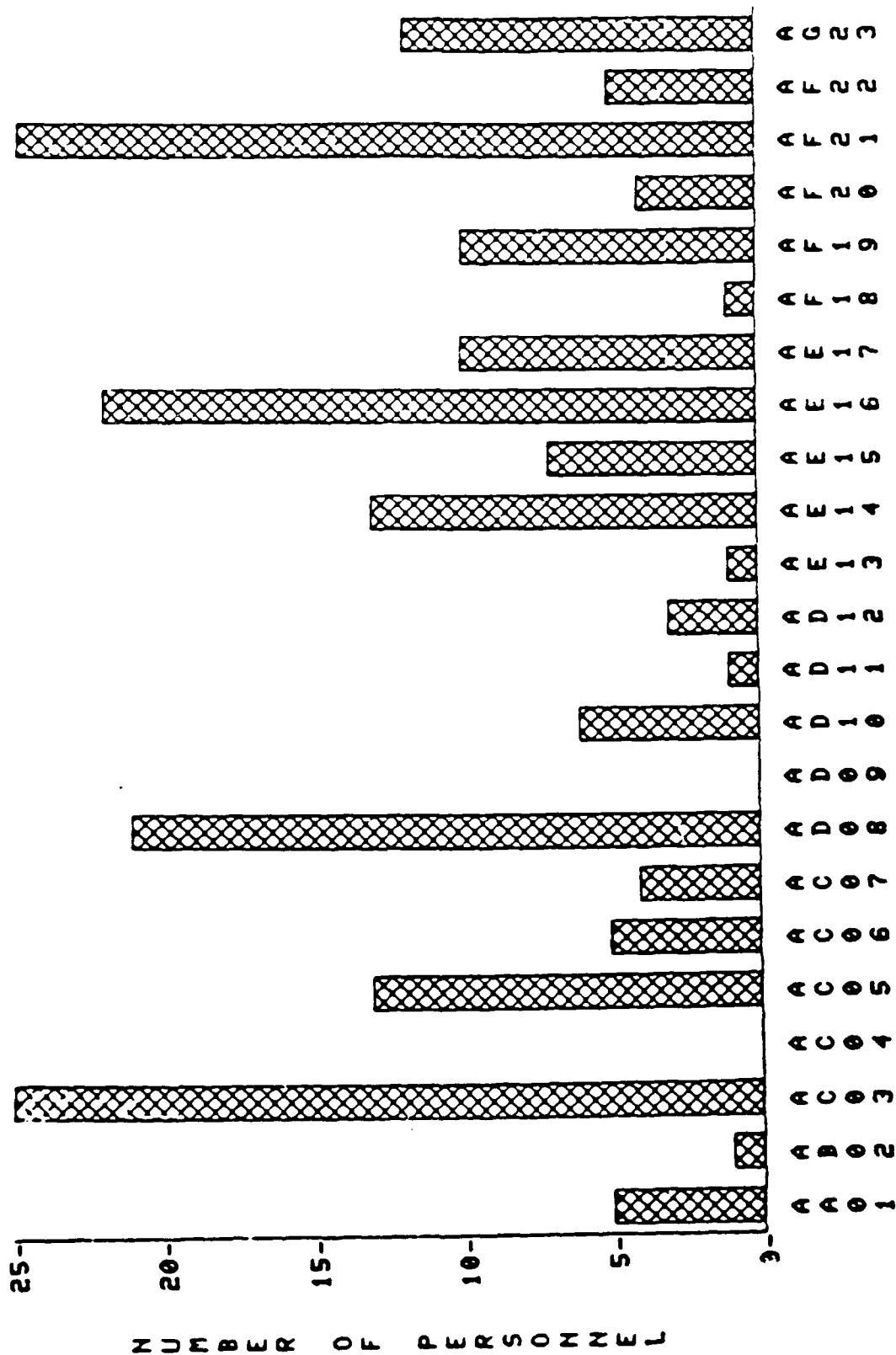
PERSONNEL STRENGTH PROFILE BY ASSIGNED MISSION
BASED ON AUTHORIZED AND ASSIGNED STRENGTHS
(GIPSY SYSTEM TEST #1)

	AUTHORIZED	ACTUAL	OFFICER	ENLISTED	STR-RATIO	MSN-READY	READINESS	SIGNIFICANCE
AA01	48	45	5	40	.9375000	3.2000000	3	1
AB02	40	31	1	30	.7750000	3.8700678	3	4
AC03	65	75	25	50	1.1538462	2.6000000	3	9
AD04	65	29	0	29	.4461538	6.7241380	3	16
AE05	75	55	13	42	.7333333	4.0900091	3	25
AF06	113	73	5	68	.6460177	3.0958904	2	36
AG07	15	23	4	19	1.5333333	.6521739	1	49
AD08	23	22	21	1	.9565217	5.2272728	1	64
AE09	35	35	0	35	1.0000000	4.0000000	5	81
AD10	40	52	6	46	1.3000000	2.3076923	4	100
AE11	50	55	1	54	1.1000000	1.8181818	3	121
AD12	105	103	3	100	.9809524	1.0194175	2	144
AE13	29	13	1	12	.4827589	6.6923077	1	169
AD14	39	29	13	16	.7435897	6.7241380	3	196
AE15	6	7	17	0	1.1666667	4.2857143	5	225
AE16	20	24	22	2	.8275862	6.0416667	5	256
AE17	40	42	10	32	.8571429	5.8333333	5	289
AF18	150	135	1	134	.9000000	4.4444444	5	324
AD19	133	100	10	90	.7518797	6.6500000	4	361
AE20	170	168	4	164	.9882353	5.8595238	5	400
AF21	50	60	25	35	1.2000000	4.1666667	5	441
AG22	88	95	5	90	1.0795455	4.6315789	5	484
AG23	115	115	12	103	1.0000000	5.0000000	5	529

UNCLASSIFIED

Figure 2-5. Tabular Report.

UNCLASSIFIED
OFFICER PERSONNEL BY ASSIGNED MISSION
CIPs, SYSTEM TEST 01



OFFICER

Figure 2-6. Bar Graph

UNCLASSIFIED ENLISTED PERSONNEL BY ASSIGNED MISSION

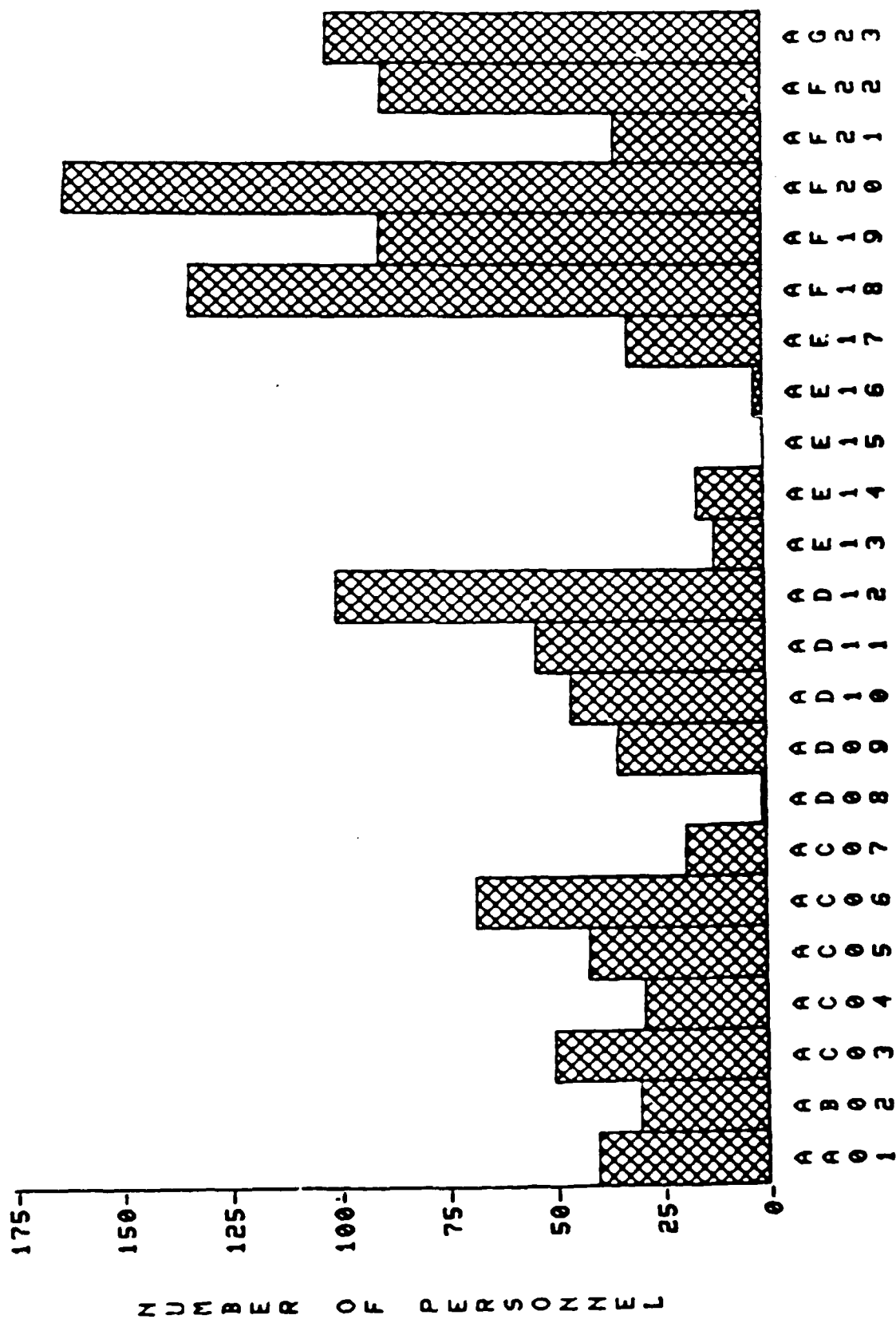


Figure 2-7. Histogram

ENLISTED

UNCLASSIFIED

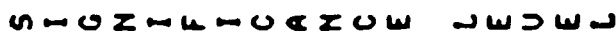
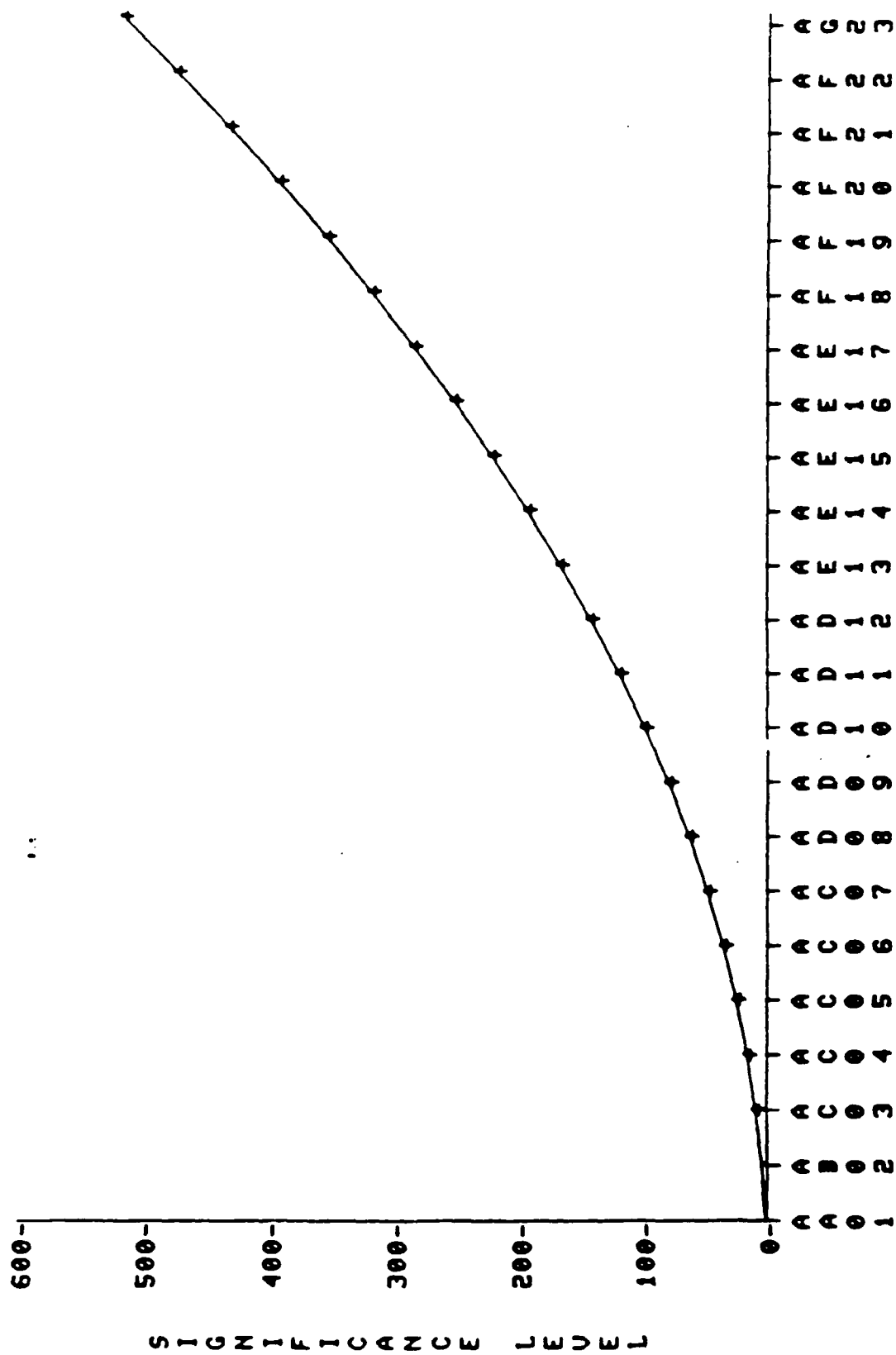


Figure 2-8. Point Graph

UNCLASSIFIED
 LIKELIHOOD OF MISSION COMPLETION RATINGS
 (MISSION DAYS)



SIGNIFICANCE

Figure 2-9. Line Graph With Single Line

Retinal Level

Strength Ratio

Strength Ratio	Retinal Level (Solid Circles)	Retinal Level (Solid Squares)
0001	1.0	1.0
0002	1.2	1.0
0003	1.0	1.2
0004	1.2	1.4
0005	1.0	1.6
0006	1.2	1.8
0007	1.4	2.0
0008	1.2	2.2
0009	1.0	2.4
0010	1.2	2.6
0011	1.4	2.8
0012	1.2	3.0
0013	1.0	3.2
0014	1.2	3.4
0015	1.4	3.6
0016	1.2	3.8
0017	1.4	4.0
0018	1.2	4.2
0019	1.0	4.4
0020	1.2	4.6
0021	1.4	4.8
0022	1.2	5.0
0023	1.0	5.2

2-18

UNCLASSIFIED
ENLISTED PERSONNEL BY ASSIGNED MISSION

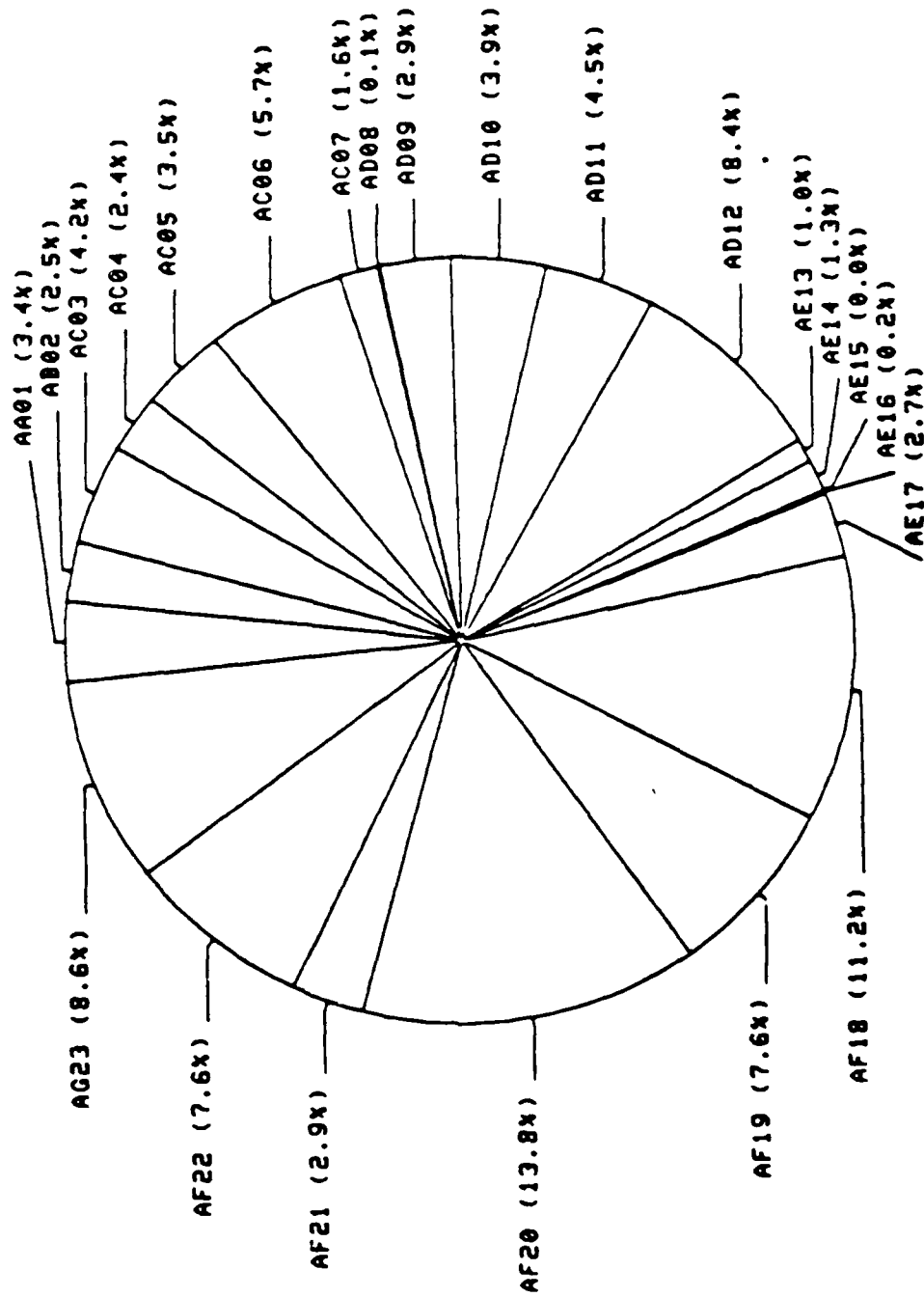


Figure 2-11. Pie Chart


```

SET SIZE JUNK.
BUILD NEW REPORT.
ASSIGN COL PCNT-OFFICER - OFFICER/ACTUAL*100.
      PCT-ENLISTED - ENLISTED / ACTUAL * 100.
      ROW ACTOTALS - AC03 + AC04 + AC05 + AC06 + AC07.
      ADTOTALS - AD08 + AD09 + AD10 + AD11 + AD12.
END ASSIGN.
END.
TITLE (SIZE L) 'ASSIGNMENT OF NEW ROWS AND COLUMNS';
      'USING BUILD NEW REPORT'.
SET SIZE MED.
DISPLAY REPORT.

```

Figure 2-12. BNR Report Modification

UNCLASSIFIED

ASSIGNMENT OF NEW ROWS AND COLUMNS
USING BUILD NEW REPORT

1935 08 AUG 86
1 OF 2

	AUTHORIZED	ACTUAL	OFFICER	ENLISTED	STR-RATIO	MSN-READY	READINESS
AA01	48	45	5	40	.9375000	3.200000	3
AA02	40	31	1	30	.7750000	3.870368	3
AA03	65	75	25	50	1.1538402	2.600000	3
AC04	65	29	0	29	.4461538	6.724138	3
AC05	75	55	13	42	.7333333	4.090309	3
AC06	113	73	5	68	.6460177	3.095090	2
AC07	15	23	4	19	1.5333333	.652174	1
AD08	23	22	21	1	.9565217	5.227273	1
AD09	35	35	0	35	1.0000000	4.000000	5
AD10	40	52	6	46	1.3000000	2.307692	4
AD11	50	55	1	54	1.1000000	1.818182	3
AD12	105	103	3	100	.9809524	1.019117	2
AE13	29	13	1	12	.482759	6.692308	1
AE14	39	29	13	16	.7435897	6.724138	3
AE15	6	7	7	0	1.1666667	4.285714	5
AE16	29	24	22	2	.8275862	6.041667	5
AE17	49	42	10	32	.8571429	5.833333	5
AF18	150	135	10	134	.9000000	4.444444	5
AF19	131	106	19	90	.7518797	6.650000	4
AF20	170	168	4	164	.9882353	5.053524	5
AF21	50	60	25	35	1.2000000	4.166667	5
AF22	88	95	5	90	1.0795455	4.631579	5
AG23	115	115	12	103	1.0000000	5.000000	5
ACTOTALS	333	255	47	208	4.5126844	17.163111	12
ADTOTALS	253	267	31	236	5.3374741	14.372564	15

UNCLASSIFIED

Figure 2-13. Modified Report

UNCLASSIFIED
ASSIGNMENT OF NEW ROWS AND COLUMNS
USING BUILD NEW REPORT

	SIGNIFICANCE	PCMT-OFFICER	PCT-ENLISTED
AA01	1	11.1111	88.8888
AB02	4	3.22581	96.77419
AC03	9	33.33333	66.66667
AD04	16	.00000	100.00000
AE05	25	23.63636	76.36364
AF06	36	6.84932	93.15069
AG07	49	17.38130	82.60870
AH08	64	95.45455	4.54545
AI09	81	.00000	100.00000
AJ10	100	11.52846	88.46154
AK11	121	1.81818	98.18182
AL12	144	2.91262	97.08738
AM13	169	7.69231	92.30769
AN14	196	44.82759	55.17241
AO15	225	100.00000	.00000
AP16	256	91.66667	8.33333
AQ17	289	23.99952	76.19048
AR18	324	.74074	99.25926
AS19	361	10.00000	90.00000
AT20	400	2.38995	97.61905
AU21	441	41.56567	58.33333
AV22	484	5.26316	94.73684
AW23	529	10.43478	89.56522
AX24	576	81.21032	418.78969
ADTOTALS	510	111.72381	388.27619

UNCLASSIFIED

Figure 2-14. Last Page of Modified Report

- h. Map (figure 2-15) - GIPSY currently has all international boundaries and coastlines defined in a map file. The Mercator Cylindrical Projection is currently used to display maps of any selected area of the world, excluding the polar regions. This is automatically scaled to the specified geographic window for maximum display area.
- i. Grid - A geographic grid showing key parallels and meridians within the geographic area can be superimposed on the geographic display. The dotted lines in figure 2-15 represent the result of requesting a grid. As the amount of geographic area changes, the grid lines are automatically recalculated in order to provide a valid frame of reference. Of course the user does have the option of specifying individual grid lines.
- j. Symbol Plot (figure 2-16) - Symbol representing units are plotted at the locations specified by coordinates in the set of records retrieved from the user's data file or selectively assigned by the user. The symbol plot can be displayed on the map with or without a track plot.
- k. Track Plot (figure 2-17) - A track plot is a geographic display in which successive data points are connected to define a path of a unit's movement, reconnaissance flight path, ship movement, etc. Any number of tracks can be generated and displayed on the map.
- l. Zoom (figure 2-18) - One can zoom in on any portion of the geographic display. The system automatically readjusts all parameters to insure a good display. The area to be zoomed in on or out from can be specified by graphic cursors rather than be predefined zooming ratios. The position of the cursor is shown by the dashed lines; the result of the zoom is shown in figure 2-19.
- m. List (figures 2-20 and 2-21) - The graphic cursors can be used to identify an area from which will be listed all data items (extracted from the file) that fall within the area. The dashed lines in figure 2-20 show the window selected, and figure 2-21 shows all the data points falling within the window.
- n. Printed Reports (figure 2-22). GIPSY provides a very effective file browsing capability which produces automatically semi-formatted reports. The GIPSY language allows the user to interactively browse through his/her data by specifying the conditions under which a record is to be displayed and then to print all records meeting that result. The qualifying data is immediately displayed.
- o. Formatted Reports (figure 2-23). The Generalized Data Reports Module (GDRMOD) provides flexible and comprehensive report generation and gives the user complete control over the format of his/her reports.

UNCLASSIFIED
GIPSY WORLD MAP FILE
WITH GEOGRAPHIC GRID

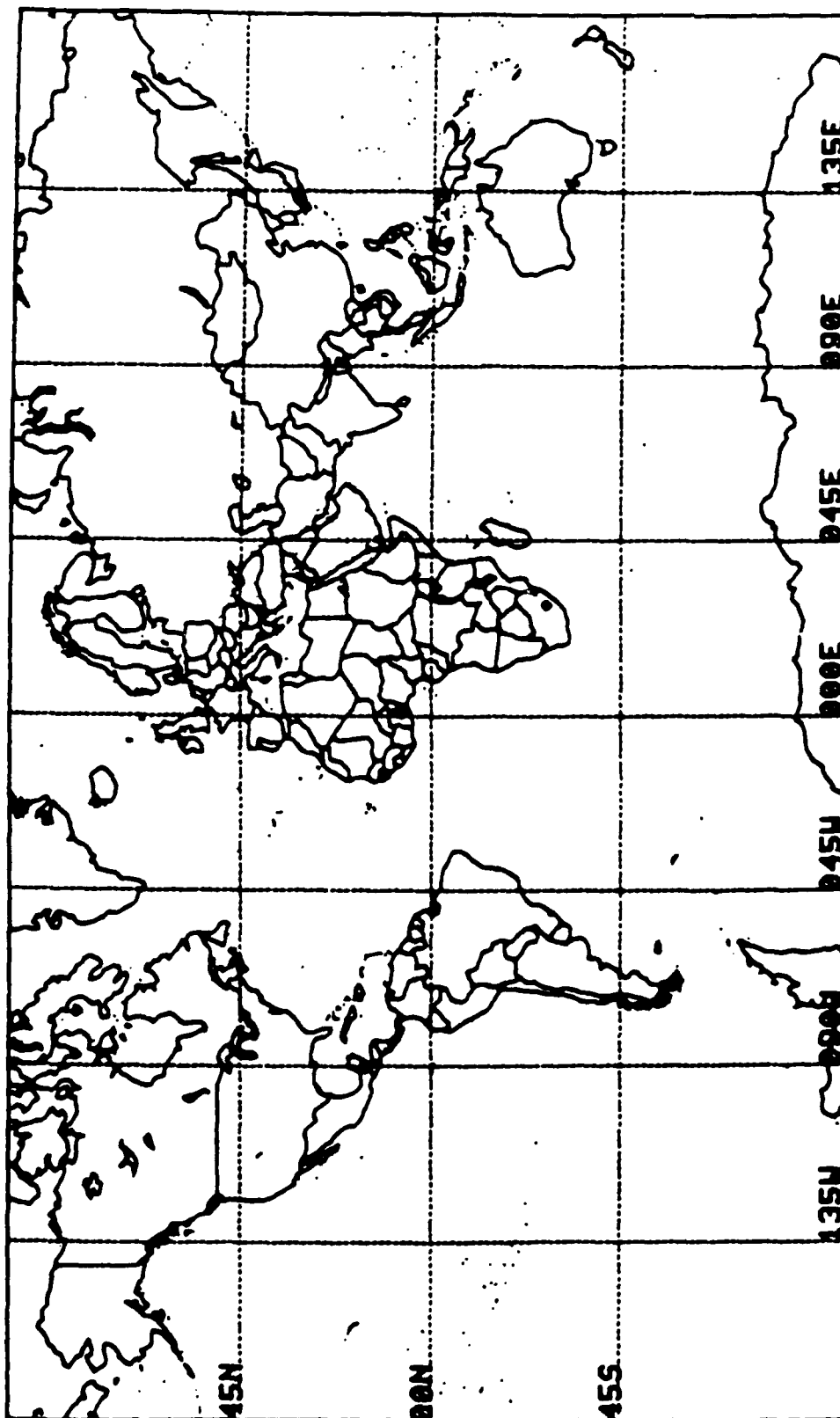


Figure 2-15. GIPSY World Map With Grid Lines

GIPSY SYMBOL PLOT

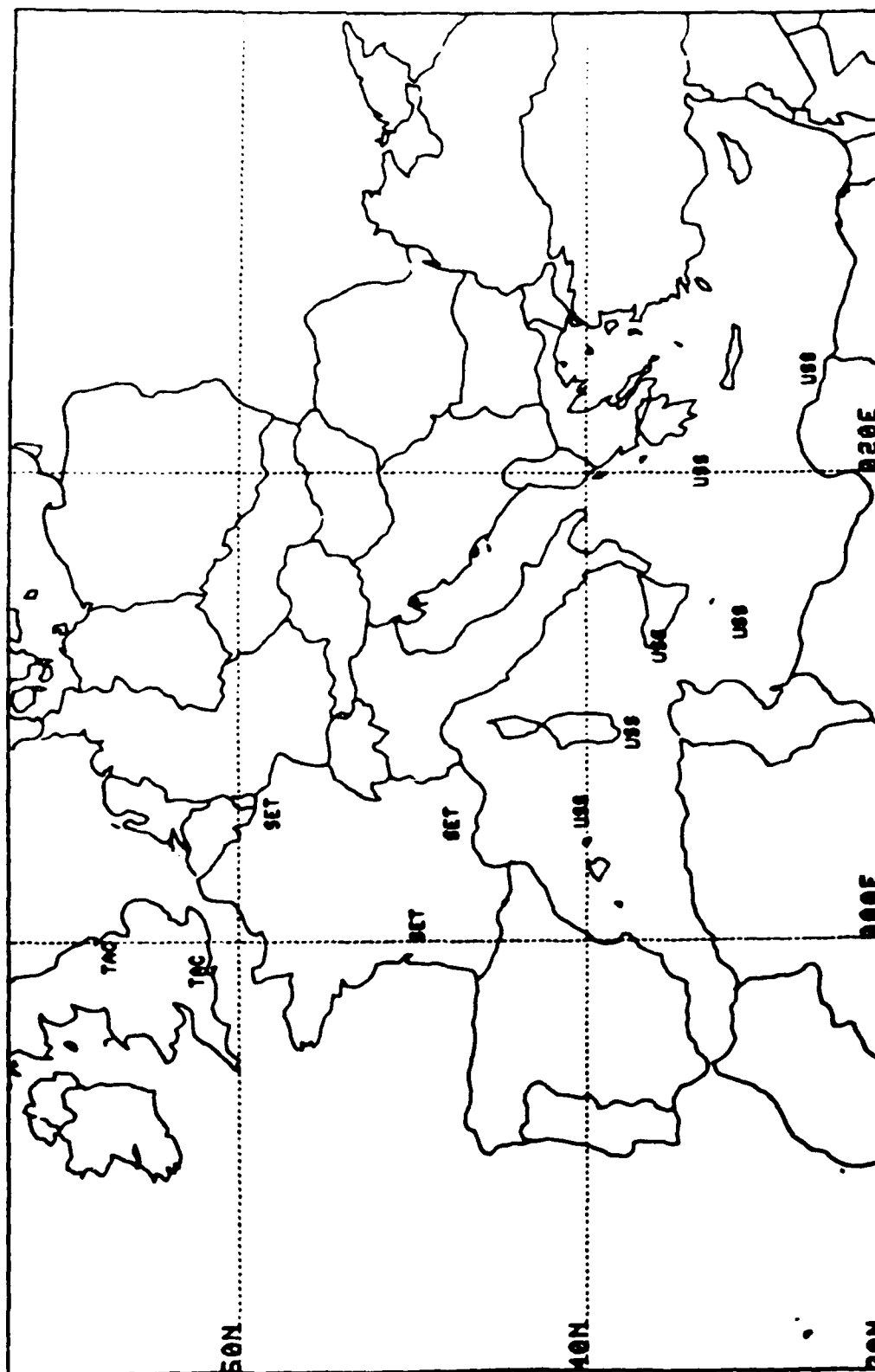


Figure 2-16. Symbol Plot

GIPSY TRACK PLOT

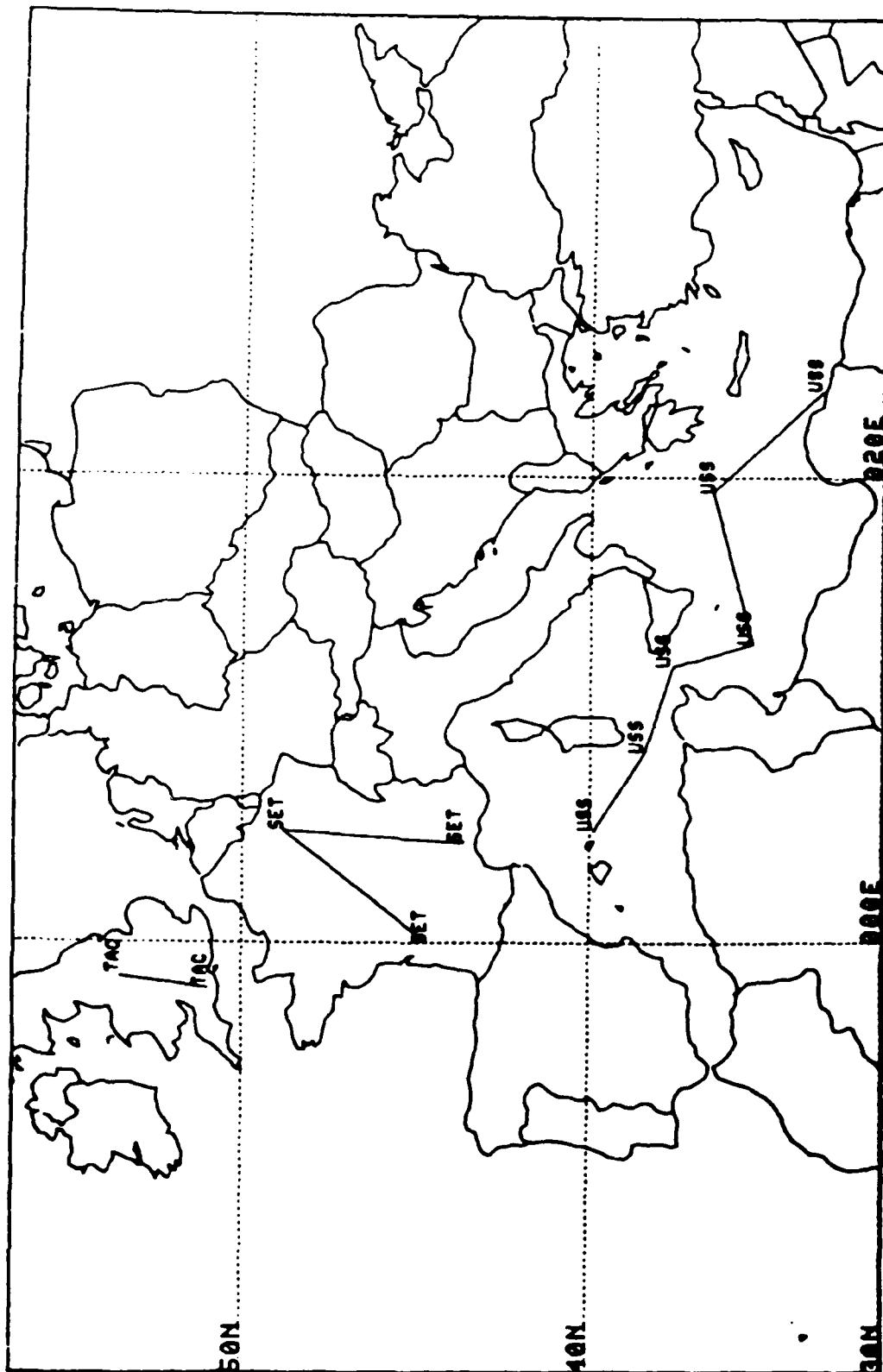


Figure 2-17. Track\Symbol Plot

GIPSY TRACK PLOT

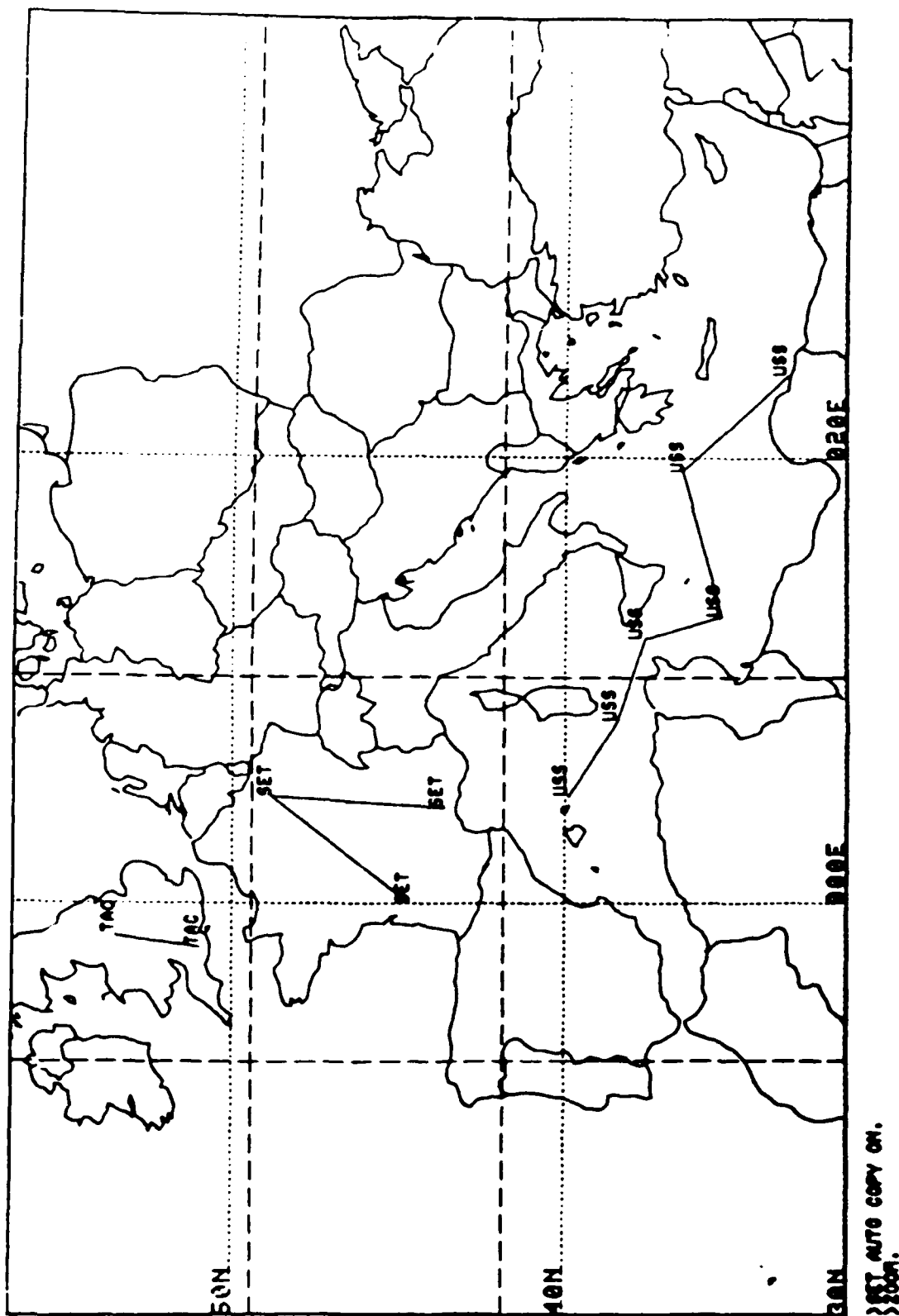


Figure 2-18. Zoom in on Track and Symbol Plot

GIPSY TRACK PLOT

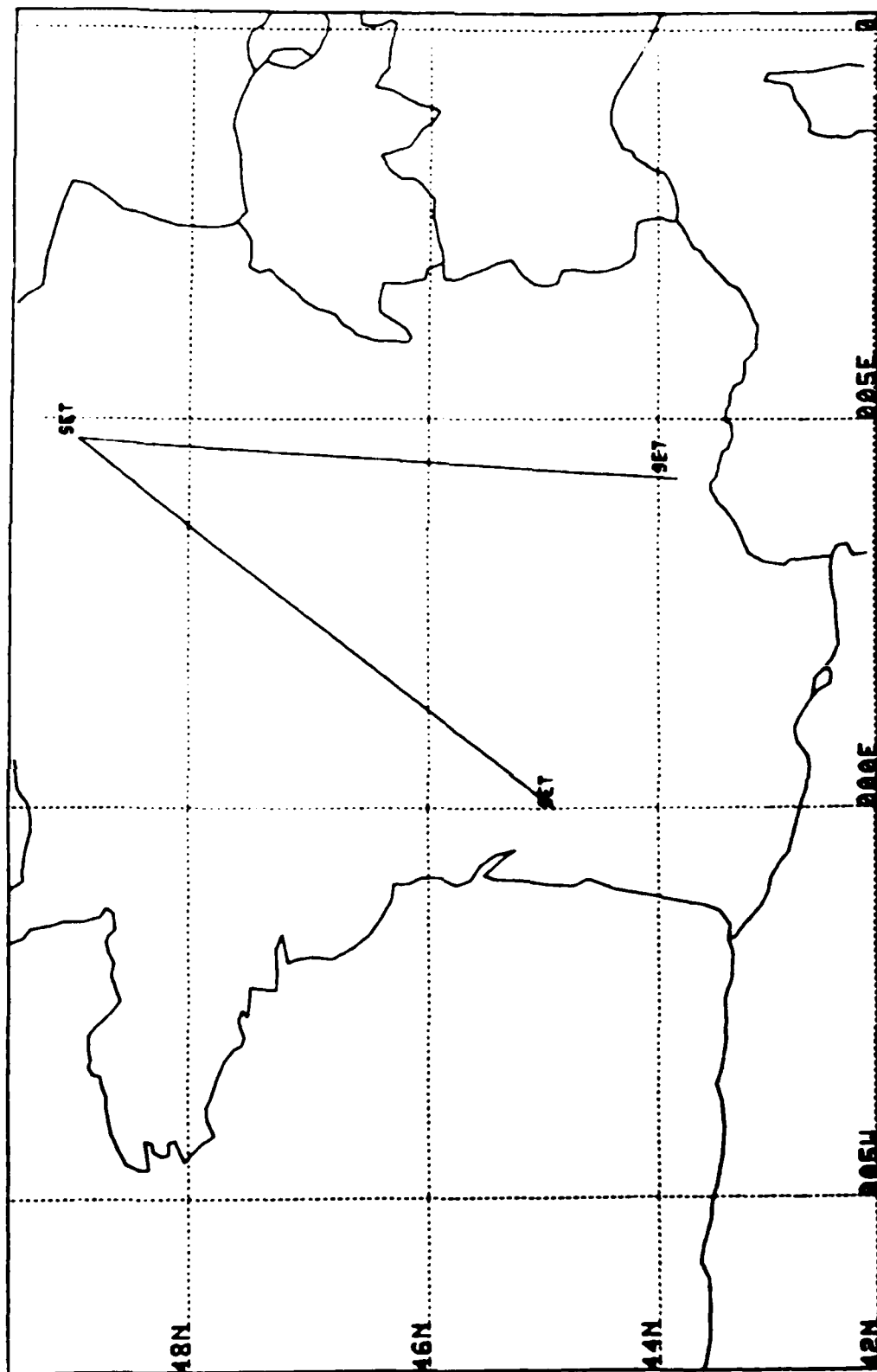


Figure 2-19. Result of Zoom

VERY UNCLASSIFIED

RECONNAISSANCE MISSIONS - CARRIBBEAN AREA

1 JULY 1981

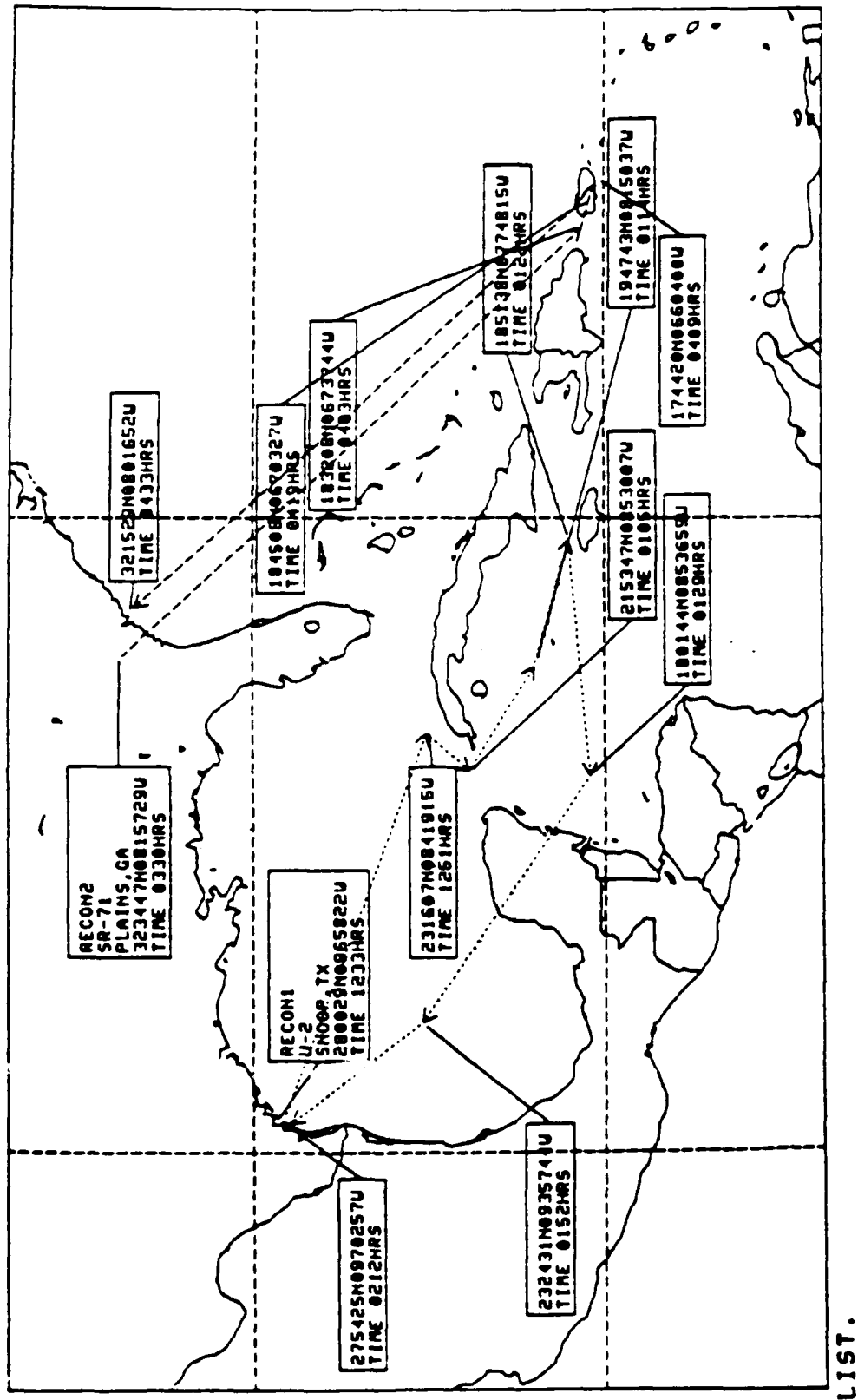


Figure 2-20. Window for Geographic List

VERY UNCLASSIFIED

RECONNAISSANCE MISSIONS - CARRIBBEAN AREA 1 JULY 1981

MISSION	LOCATION	POINT	AIRCRAFT	DRIGIN	REPTIM
RECON1	280029N0965822U	280029N0965822U	J-2	5N00P, TX	TIME 1233HRS
RECON1	231607N0841915U	231607N0841915U	J-2	5N00P, TX	TIME 1251HRS
RECON1	215347N0853007U	215347N0853007U	J-2	5N00P, TX	TIME 0105HRS
RECON1	194743N0815037U	194743N0815037U	J-2	5N00P, TX	TIME 0114HRS
RECON1	185138N0774815U	185138N0774815U	J-2	5N00P, TX	TIME 0122HRS
RECON1	180144N0853659U	180144N0853659U	J-2	5N00P, TX	TIME 0129HRS
RECON1	232431N0935744U	232431N0935744U	J-2	5N00P, TX	TIME 0152HRS
RECON1	275425N0970257U	275425N0970257U	J-2	5N00P, TX	TIME 0212HRS

Figure 2-21. Geographic List

MISSION	AIRCRAFT	LOCATION	ORIGIN	POINT	REPTIM
RECON1	U-2	280029N0965822U	SN00P, TX	280029N0965822U	TIME 1233HRS
RECON1	U-2	231607N0841915U	SN00P, TX	231607N0841915U	TIME 1251HRS
RECON1	U-2	215347N0853007U	SN00P, TX	215347N0853007U	TIME 0105HRS
RECON1	U-2	194743N0815037U	SN00P, TX	194743N0815037U	TIME 0114HRS
RECON1	U-2	185138N0774815U	SN00P, TX	185138N0774815U	TIME 0122HRS
RECON1	U-2	180144N0853659U	SN00P, TX	180144N0853659U	TIME 0129HRS
RECON1	U-2	232431N0935744U	SN00P, TX	232431N0935744U	TIME 0152HRS
RECON1	U-2	275425N0970257U	SN00P, TX	275425N0970257U	TIME 0212HRS

Figure 2-22. Sample Printed Report

AIRFIELDS ALPHABETICAL INDEX
A - G

	A	B	C	D	E	F	G
ALABAMA	2	2	2	2		2	2
FLORIDA	4	4	4	4	4	4	4
GEORGIA	8	8	8	8	8	9	9
MISSISSIPPI	11	11	11			11	11
SOUTH CAROLINA	13	13	13	13		13	13

Figure 2-23. Sample Formatted Report (Part 1 of 2)

AIRFIELDS ALPHABETICAL INDEX

H - H

	H	I	J	K	L	M	N
ALABAMA	2		5	5		2	5 9
FLORIDA	4		9		5	5	9
GEORGIA	9				9	9	
MISSISSIPPI	11		11	11	12	12	
SOUTH CAROLINA	13				13	13	

Figure 2-23. Sample Formatted Report (Part 2 of 2)

THIS PAGE INTENTIONALLY LEFT BLANK

SECTION 3. STAFF FUNCTIONS RELATED TO TECHNICAL OPERATIONS

The purpose of this section is to provide a detailed description of how to use GIPSY in an on-line interactive environment. This will be accomplished using a combination of instructions, discussions, and examples. Those users who have no need for statistical or management graphs may safely skip section 3.3. Those users who have no geographic data or interest in geographical displays may skip section 3.4.

3.1 Initiation Procedure

The initiation procedure described here includes the entire standard TSS log-on sequence. The user master catalog (UMC) shown in examples in this document is the one defined at the JDSSC. The UMC is the first eight characters of the catalog/file string which precedes the first slash (/) in the catalog/file string. Each H6000 user must have a USERID and password in order to perform any task on the computer. Your local site must provide these for you. To access the H6000 the user must first ensure that the device is on and that the computer is operational. You must now get the computer's attention. The method of accomplishing this will vary from device to device and site to site. These procedures should be documented by the site. This process is not a function of GIPSY but of the H6000 log-on process. Typical log-on sequences are shown in figure 3-1.

When the log-on process is complete, a system prompt "*" appears. GIPSY may be initiated by responding "GIPSY". There are several statements which may be included on the GIPSY command line. These statements will be discussed later. When the command is received, GIPSY will respond with a ready message and release number such as:

GIPSY Release 5.1 (30 APRIL 86) ***PRODUCTION SYSTEM***

When the symbol ">" appears, GIPSY is ready to accept your input request. We call this symbol a prompt character. The prompt character indicates that GIPSY is ready for keyboard input. If you transmit a null line (e.g., carriage return only), GIPSY will respond with a message telling you what it is expecting as input. This is true at any time except in error correction or in answer to a specific question (in an error correction sequence, a null line deletes the entire statement containing the error). When the display screen contains a data display the audible alarm (bell) will sound to indicate that GIPSY is ready for keyboard input.

3.2 GIPSY Language: Syntax and Semantics

Before getting into the composition rules, let us first define the terms "syntax" and "semantics" and define our meta-language. When we use the word "syntax" we are referring to the grammatical or technical construction of a statement. "Semantics" are the meanings associated with a word, phrase or sentence.

\$5,TSS

PLEASE LOG IN.

L -PER DJ8XI344XX -PIC XXXXXXXXX -PJ DJ8XI344XX -SCC UZZ

0316000

PROGRAM NAME -TSS

PLEASE LOG IN.

L -PER DJ8XI344XX -PIC XXXXXXXXX -PJ DJ8XI344XX -SCC UZZ

Figure 3-1. VIP 7705 and Tektronix 4014 Log-on Procedures

The symbols defined here are widely used and many are well known. However, they are repeated here to eliminate the possibility of confusion.

Metasymbol

Semantics

[]

One item of information enclosed in brackets may be selected; the information may be omitted entirely and system defaults will apply. Where appropriate, the selected default is underlined.

{ }

A list of items will be contained within the braces; one item must be selected from the list.

] ⁿ }

A subscript/superscript or brackets and braces indicates the number of choices which may be made.

GIPSY

Items shown in uppercase are system-recognized words in the GIPSY vocabulary and should be entered as shown. All words in the GIPSY vocabulary are unique up to six characters; consequently, only the first six characters are validated; remaining characters are ignored but are included for readability.

< filename>

Items shown in lowercase bounded by carets are either user-defined elements or a single generic term to be expanded later in the discussion. These are separately discussed in the text.

. or . . .

The ellipsis is used to indicate that repetition of the immediately preceding information is allowed.

3.2.1 Composition Rules. GIPSY follows many of the rules of the English language. Punctuation in the form of spaces, periods, commas, and semicolons is part of the language and must be entered as shown in the syntactical formats of GIPSY statements. There is a degree of freedom permitted in punctuation if there is no possibility of ambiguity. These cases will be addressed as each statement is discussed. As a general rule, every statement must end with a period, and all items in a list must be separated by commas. Spaces are significant in that, as in English, they terminate a word. The semicolon is used to separate multiple clauses in a single sentence, such as an arithmetic expression followed by a conditional expression.

There are some ambiguities in the English language that we should take note of in looking at punctuation symbols. What do you see as the difference between a period and decimal point? A hyphen and a minus sign? The way you tell them apart is by observing the context in which they appear. GIPSY too looks at

the context but it cannot make all the adjustments that you mentally make in deciding whether the symbol "-" is a minus sign or a hyphen.

The context rules which GIPSY uses in distinguishing these two symbols are as follows:

- a. If the symbol "-" is embedded within a string of characters, it is a hyphen (e.g., START-TIME).
- b. If the symbol "-" is preceded by a nonalphabetic, nonnumeric character (e.g., blank, comma, etc.) it is a minus sign (e.g., (PRICE - COST)).
- c. The symbol "." is a decimal point if and only if it is followed immediately by a numeric string. In all other cases it is considered a period.

There is one primary rule to follow when putting together a series of GIPSY statements: all field names or other item names must be known to GIPSY prior to their use. In other words, you must define all field names in a FDT to GIPSY before using the fieldname in the PCS; you must define your equation before you attempt to reference it by name.

The rules for defining field names are discussed in section 3.2.5. However, there is one concept which should be discussed here -- the concept of partial field notation. Partial field notation allows you to reference part of a data field rather than the whole data field. Assume that you have a field called DATE which contains dates in the form 26 FEB 55 but for some reason you have only a need to look at the month. You could get to the month in either of two ways: first, redefine the FDT to add a field name for month within the DATE field; or second, you could ask GIPSY for the data in character positions 4 thru 6 in the DATE field. This would be accomplished by specifying DATE (4/6). The character positions are always relative to the start of the field and must always be specified in parentheses. The specific syntax is:

`<fieldname> [(<start position> [/ <end position>])]`

The <start position> and <end position> must be unsigned integers, less than or equal to the length of the field, designating the character positions within the field being referenced. If no <end position> is specified, a length of one character is assumed. Any nonbinary type data is subject to partial field notation, but the partial field (selecting positions) will be processed as alphanumeric. Care must be taken not to go beyond the limits of the data field when asking for a partial field. In any statement where a field name is used, the field can have partial field notation.

The types of data recognized by GIPSY are alphanumeric, integer, floating point (character form with or without decimal point), coordinate, logical, binary integer, and binary floating point. GIPSY provides automatic

conversion of the data types to match the form needed for comparisons and computations. For example, a data field typed as alphanumeric may be used in calculations if the contents of the selected field are numeric.

GIPSY remembers each statement that is input. Whenever a statement is syntactically correct and complete, it is saved for the duration of the GIPSY session unless GIPSY is told not to save it. Certain statements, however, are logically not remembered; instead, their results are recorded. Statements not automatically saved are noted in the discussion of the language input statements in section 3.2.3.3; otherwise all statements are recorded on a temporary Process Control Statement (PCS) file.

There are no card image or end-of-line requirements except as noted. That is, any statement may be continued on as many lines as necessary to define the entire statement. GIPSY will continue to accept input as belonging to a single sentence until a period is encountered.

When all report descriptions are complete and you are ready to begin graphic or geographic displays, the single command RUN must be entered.

GIPSY's error correction tools are designed to avoid unnecessary and redundant typing. If a syntax error is encountered the input containing the error will be redisplayed along with a set of error pointers which identify the word or string of characters causing the error condition. Correction is made by typing in a replacement for the information identified by the error pointers. Do not retype the entire line; retype only the indicated information. All information to the left of the error pointers is syntactically correct and has already been accepted. Consider the following example in which an invalid size is specified

SET SIZE BIGG.

GIPSY would respond with the error sequence

SET SIZE BIGG.
 ^ ^

Invalid character size. Allowable character sizes are JUMBO, LARGE, MEDIUM, OR SMALL. Enter correction or a null line to delete the entire statement.

Note the symbol " ^ " is used as the error pointer marking the range of the data causing the error condition. This sequence is asking for a replacement of the characters "BIGG". GIPSY will then reinitiate the syntax checking with the first character or the replacement input that was inserted between SIZE and the period.

If the user response to this error prompt is:

JUMBO

then the resulting statement would be:

```
SET SIZE JUMBO.
```

The replacement information need not agree in length or structure with the information it is replacing. The only restriction is that the replacement cannot cause the resulting input line to exceed 133 characters.

If it is desired that the entire line should be deleted rather than attempting to correct it, a null line (e.g., carriage return with no information) will cause the entire statement to be deleted and a new prompt character displayed.

GIPSY syntax checking is interactive; consequently, a syntactically correct statement cannot be deleted. However, in most cases it can be replaced by reissuing the statement with corrections applied. If you are in the middle of a syntactically correct statement and you desire to cancel the statement, a deliberate error will cause an error prompt at which time the statement can be deleted. Observe the following errors and correction sequences. Note the results following each //LIST 1 command:

```
>RENAME COLS ABLE THRU CHARLEI PREFIX "X-1".  
Column vector not found. (248)
```

```
RENAME COLS ABLE THRU CHARLEI PREFIX "X-1".  
                  ^          ^
```

```
>CHARLIE  
>//LIST 1  
RENAME COLS ABLE THRU CHARLIE PREFIX "X-1".
```

Catalog-file descriptors (cfd) require special attention in the GIPSY language. The end of sentence period is generally omitted on statements which end with a cfd to avoid confusion with the period which may be part of the cfd.

In handling catalog file descriptors, GIPSY follows the convention that it will remove any file it attaches from the AFT. It will not remove any file it finds already in the AFT. Files should be referenced using the full cfd rather than just the file name. This will allow file pointers retained by GIPSY to be complete in the event that a data structure referencing that file is saved.

3.2.2 GIPSY Command Options. The nonprogrammer GIPSY user may safely skip to section 3.2.3 without loss of continuity. This section is oriented toward the applications programmer who may be assisting the user analysts in their GIPSY application.

When GIPSY is initiated by responding GIPSY, GIPSYD or GIPSYG to a system prompt, several options are available. These options allow the user to enter the statistical or geographic display module directly, to predefine files,

specify an alternative version of the GIPSY system and provide additional or replacement modules for system execution. The average user will use only a couple, if any, of the options; however, it may be useful to be aware that they are available. The full syntax of the GIPSY command statement is:

$\left\{ \begin{array}{l} \text{GIPSY} \\ \text{GIPSYD} \\ \\ \text{GIPSYG} \\ \text{GIPSYR} \end{array} \right\}$	<u>PRODUCTION</u>	$\left[\begin{array}{l} \text{DAFC <cf>} \\ \text{CUELIB <cf>} \\ \text{GCFILE <cf>} \\ \text{NO-MSG} \end{array} \right]$	4	$\left[\begin{array}{l} \text{PCS <cf>} \\ \text{FILE <cf>} \\ \text{GDS <cf>} \\ \text{QDF <cf>} \end{array} \right]$	4	[module <cf>]
	OPTEST					
	DEV		0	0		

If more than one option is used each must be separated from the other by a semicolon; the statement may be continued onto following lines by terminating the line with a semicolon. The cfd is the catalog file descriptor specifying the file for the option. These keywords are order dependent. The keywords on the command have the following definitions:

GIPSY This command initiates GIPSY from the top of the system to build the graphic structures for either graph and reports or geographic displays or both.

GIPSYD This command initiates the GIPSY system and skips directly to the graph and report display module. You must already have a GDS or plan to key in the data to build it.

GIPSYG This command initiates GIPSY and transfers directly to GIPSY the geographic output display module. This method of entry bypasses all GIPSY data file accesses. You must already have saved a QDF/QDT, or plan to display geographic information not associated with a data file retrieval.

GIPSYR This command initiates GIPSY's Output Processor. Inputs to this module can be a saved QDF/QDT and/or user specified BOOKFILE.

DAFC <cf> Specifies a previously saved Directive Action and File Control (DAFC) file. See section 3.2.3.8, the Directive Action and File Control (DAFC) file, for a more detailed description of this file.

CUELIB <cf> Used to override the standard cue library. The cue library contains all GIPSY messages and system run stream parameters.

PRODUCTION Selects the version of the system to be executed. The
 OPTEST default is to use the production system that is on your system library. OPTEST specifies an operation test version. The OPTEST option may not be available at all sites. Information specific to each system is maintained on that

system's CUELIB. When used, the system version must be specified before any replacement modules are given.

- GCFILE <cf> Specifies an ASCII command list parameter file. Any GIPSY command option may be placed on the GCFILE; order, dependency and syntax of the options must be maintained as if they were typed on GIPSY initialization command. If the option list on the GCFILE ends with a semicolon, the system will return to the terminal for continuation of the GIPSY command line options.
- NO-MSG Prevents the log-on message from being displayed when GIPSY is initiated.
- PCS <cf> Specifies a Process Control Statement (PCS) input file. See PCS statement description, section 3.2.3.3. Note - if the PCS option is used with GIPSYD command, GIPSY assumes a Graphic Data Set (GDS) will be accessed by the PCS (see section 3.3.3.8) before any displays are requested. GIPSY will not ask for a GDS if a PCS is specified on the GIPSY command .
- FILE <cf> Specifies the user data file. See data file description, section 3.2.4.1.
- GDS <cf> Specifies a user file which will be used as the Graphic Data Set (GDS). For additional information, see the GDS statement description in section 3.2.3.6 (Note: with the GIPSYD command, the GDS will be used for input to load a tabular report; with the GIPSY command this option specifies where the GDS will be written).
- QDF <cf> Specifies a user file which will be used as the Qualified Data File (QDF). See QDF statement description, section 3.2.3.7.
- module <cf> This set of parameters allows any of GIPSY's six modules to be deleted or replaced, and allows for the insertion of a user module before and/or after each GIPSY module. The standard run stream may be modified by replacing the GIPSY module with an alternate module. The prefix PRE- or POST- appended to the module name will cause the specified user module to be executed prior to or after the indicated GIPSY module, respectively. Additionally there is a PRE-MOD which will be executed ahead of all modules. A GIPSY module can be deleted by specifying the module name without a catalog file descriptor. The GIPSY module indicators are as follows:

SYNTAX [<cf>]	The language syntax module which handles all language statements up to and including the RUN statement
DATSEL [<cf>]	The data retrieval module
MTXGEN [<cf>]	The module which builds the tabular report
DISPLA [<cf>]	The reports and graphics display module
GEOMOD [<cf>]	The geographic display module
GDR	The Generalized Data Reports Module is used to produce user specified textual reports. See section 6 for details on this module.

The full definition of <module overrides> syntax is:

```

PRE-MOD [<cf>]
PRE-SYNTAX [<cf>]
SYNTAX [<cf>]
POST-SYNTAX [<cf>]
PRE-DATSEL [<cf>]
DATSEL [<cf>]
POST-DATSEL [<cf>]
PRE-MTXGEN [<cf>]
MTXGEN [<cf>]
POST-MTXGEN [<cf>]
PRE-DISPLA [<cf>]
DISPLA [<cf>]
POST-DISPLA [<cf>]
PRE-GEOMOD [<cf>]
GEOMOD [<cf>]
POST-GEOMOD [<cf>]

```

3.2.3 Classification, Title, and Auxiliary Functions. The statements discussed under this heading set up the operating environment in terms of defining classification markings, titles, current character size, GIPSY input source statement location and supporting files.

3.2.3.1 Classification Marking. A classification statement specifies the classification markings which are to appear on each report. This marking will also appear at the top of the display screen after GIPSY clears the screen. The syntax of the classification statement is:

```

CLASS [(SIZE <size option>, COLOR <color option>, TOPBOT)]
      <classification code> " <textual classification> ".

```

Size specification may occur either before or after the classification caveat. The <size option> identifies the desired size of the text displayed, and may be any of the following:

JUMBO J

MEDIUM MED M

LARGE L

SMALL S

If the requested character size does not exist on the device on which the classification will be displayed, the size parameter will be ignored. If the user is on a color terminal, the color option specifies the color in which the classification will be displayed. If on a non-color terminal, the statement will be ignored. The TOPBOT option will display the classification at the top and bottom of the page. The classification caveat may be either a string of characters enclosed in quotes or one of the following classification codes to be translated as shown:

<u>Codes</u>	<u>Displayed Classification</u>
ZZZ	None
UZZ, UNC, U	UNCLASSIFIED
CZZ, CONF, C	CONFIDENTIAL
SZZ, SEC, S	SECRET
TZZ, TS, T	TOP SECRET

The classification will appear at the top and bottom of each page of a tabular report and at the top of graphic displays. If the TOPBOT option is specified the classification will appear at the top and bottom of graphic displays. If no size is specified and the device on which the display is generated has multiple sizes, the largest character size is used. The classification becomes a part of the graphic data set (GDS), so that if a GDS is saved and recalled at a later time, the proper classification remains associated with it.

The CLASSIFICATION statement may be reissued at any time to alter the caveat. The following are valid classification statements:

```
CLASS U.  
CLASS UZZ.  
CLASSIFICATION (SIZE SMALL) "FOR OFFICIAL USE ONLY".  
CLASS "TOP SECRET" (SIZE L).  
CLASS TZZ (SIZE JUMBO).  
CLASS (SIZE S) UNC.
```

If all the above statements were entered in the sequence shown the reports would come out marked UNCLASSIFIED. Particular attention must be paid when using literal strings as a classification. The literal string must always be enclosed in quotes. Classification codes must not be enclosed in quotes.

3.2.3.2 Report Titles. The TITLE statement allows the specification of a multi-line, multi-size heading for the GIPSY outputs. Each line is independently centered on the line at the top of the page.

The syntax is:

```
TITLE [ (LINE <line number>, SIZE <size option>, COLOR <color option>)]  
" <first line of title> " [ ; [(LINE <line number>, SIZE <size option>,  
COLOR <color option>)] " <next line of title> " ]...
```

The LINE option allows the user to specify the line number for each line of title. Blank lines will be inserted automatically into the lines of title if required.

SIZE and COLOR specifications follow the same syntax and semantics as in the CLASSIFICATION statement in section 3.2.3.1. These options are placed before the text of the line of title. The line of title is entered next enclosed in quotes. Multi-line/multi-size headings can be generated by repeating the title specification (excluding the word TITLE and the period) as often as desired. Each repetition of the title specification must be separated with a semicolon. The statement must always end with a period. If a character size is not specified, the size will default to the currently active character size. If a size is specified on one line, all succeeding lines will have the specified character size until another size specification is encountered.

The TITLE command to the title on the tabular report in figure 2-5 was:

```
TITLE  
  (SIZE LARGE) "PERSONNEL STRENGTH PROFILE BY ASSIGNED MISSION";  
  (SIZE M) "BASED ON AUTHORIZED AND ASSIGNED STRENGTHS";  
  (SIZE SMALL) "(GIPSY SYSTEM TEST NO. 1)".
```

3.2.3.2.1 Modifying Title Lines. The MODIFY TITLE statement permits the replacement of a single line in a multi-line title. The syntax is:

```
MODIFY TITLE (LINE <line number> [,SIZE <size option> , COLOR <color  
option>]) " <line of title> ".
```

The LINE specification must be included in the statement and must precede the text of the replacement line. The SIZE and COLOR specifications are optional.

The MODIFY TITLE statement can also be used to add line to an existing title. Blank lines will be automatically inserted if required.

3.2.3.3 Language Input. GIPSY language statements are called Process Control Statements (PCS). GIPSY statements can be input either through the keyboard or from a user file. The PCS statement is the vehicle through which you tell GIPSY to start accepting the language statements from a user file. The syntax is:

PCS { <cat/file string> }
 *

If an asterisk is supplied instead of a cat/file string GIPSY will use the contents of the TSS current file as your PCS input; if a cat/file string is supplied, the contents of the file will be used.

The specified file is immediately detached after its contents are copied to work space. Consequently, the specified file name does not remain in the TSS Available File Table (AFT), unless it was attached by some application other than GIPSY. This statement, and all statements in which a cat/file string is specified should have no other statement following on the same line. Note that it is possible to have an imbedded period within the cat/file string possibly creating an ambiguous situation. The cat/file string must be entirely contained on a single line of input.

When the PCS statement is encountered, GIPSY automatically switches over to the specified cat/file string for the next input. It will stay in this mode until a RETURN statement is encountered, another PCS statement is encountered, or all statements on the PCS file(s) are processed, in which case GIPSY will revert back to the terminal console input mode. If there are errors in the PCS file, GIPSY will allow them to be corrected via the standard error correction procedure. The system automatically returns to processing with the next line in the PCS file. However, if the statement containing the error is deleted, PCS file processing is terminated and the system reverts to keyboard input. A RETURN statement will resume PCS processing at the point of interruption.

The PCS file may be either an ASCII or BCD file. Line numbers are optional; if supplied on the PCS they will be automatically stripped off prior to being syntax checked. The file itself will not be altered unless the PCS is resaved on that file. GIPSY neither creates nor interprets line numbers. To allow the PCS file to be built and maintained by other line number oriented processors line numbers are permitted if desired. Note, however, that line numbers cannot be typed into GIPSY, they only occur on prestored PCS files.

If the PCS file contains a PCS statement, the contents of the new PCS file effectively replaces that particular PCS statement. There is no limit to the depth of nesting of PCS files as long as they are not used recursively. The PCS file may contain all statements needed for an entire run - including the graphic display commands. Figure 3-2 shows the use of a PCS statement. The only statement input was the PCS statement -- the one with the prompt character showing.

Every statement which is complete and syntactically correct is recorded in a data list called a temporary PCS. This will allow all input statements to be saved in user space for later recall. A PCS statement, however, does not get recorded onto the temporary PCS; all statements extracted from the PCS file are recorded instead.

```

GIPSY
GIPSY RELEASE 4.2.1 (13 MAY 85) **PRODUCTION SYSTEM**
For any problems or questions about this release contact the
GIPSY SUPPORT OFFICE AT (202) 695-3519, A/V 225-3519.
>PCS 837IDPX0/GIPSY/PCS/PCSFIL
SET SIZE LARGE.
FILE 837IDPX0/GIPSY/DATA/..TSTFIL
FDT 837IDPX0/GIPSY/DATA/..TSTFDT
SET AUTO COPY ON.
SET AUTO DTG ON.
TITLE
  (SIZE JUMBO) *PERSONNEL STRENGTH PROFILE BY ASSIGNED MISSION*;
  (SIZE L)      *BASED ON AUTHORIZED AND ASSIGNED STRENGTHS*;
               *((GIPSY SYSTEM TEST 81))*(SIZE SMALL)).

CLASS UZZ.
MATH TABLE.
  SRATIO - ACTLSTR/AUTHSTR.
  MSNRDY - (AUTHSTR/ACTLSTR)*READINESS.
END.
RETRIEVE IF MISSION(1/1) EQ A AND READINESS > 0 AND
LOCATION(7/7) - 'N' AND UNITTYPE NE ' ' AND
(INFODATE(5/6) BT 01/05 OR INFOTYPE-CURRENT).
BUILD TABULAR REPORT.
ROWS USE MISSION.
COLS *
  AUTHORIZED - SUM AUTHSTR.
  ACTUAL - SUM ACTLSTR.
  OFFICER - SUM OFFICER.
  ENLISTED - SUM ENLISTED.
  STR-RATIO - SRATIO; IF AUTHSTR NE 0.
  MSN-READY - MSNRDY; IF ACTLSTR NE 0.
  READINESS - SUM READINESS.
  SIGNIFICANCE - SUM MISSION(3/4)*MISSION(3/4).
END.
RUN

```

Figure 3-2. Sample of PCS Statement

3.2.3.4 Changing Language Input Modes. The RETURN statement allows one to return to either the PCS file mode of input or return to the terminal mode of input from the PCS. The statement syntax is:

RETURN TO { PCS
 TERMINAL } .

RETURN TO PCS will cause GIPSY to resume input from the PCS file at the point of departure. RETURN TO TERMINAL will cause a planned detour to the terminal keyboard for input to insert statements or to respecify an existing statement. When all statements in the PCS are gone, GIPSY automatically returns to terminal keyboard for input.

The prompt character ">" will be displayed if GIPSY is expecting keyboard input. The single exception is when a display is completed. In this case GIPSY rings the bell to signify completion of the display. A null entry allows GIPSY to resume.

It is possible to inadvertently cause GIPSY to return to the terminal keyboard for input due to an error in a PCS file statement that was not corrected, or by inputting a command, rather than a null entry when a completed display is on the screen. In such cases, the interrupt command //RETURN should be entered to allow resumption at the point of departure. The interrupt command performs the action but does not cause the statement to be recorded on the temporary PCS. This allows the validity of the PCS written out by the SAVE command to be retained.

3.2.3.5 Operating Environment Attributes. The SET statement is used to control the environmental attributes associated with a given terminal session. It is used to control the character size of the displayed data, automatic hard copy functions, display of prestored process control statements, etc.

The SET statement causes the indicated attribute to be set up immediately. The setting of that attribute remains in effect until it is respecified.

The syntax forms for this statement are:

(1) SET [AUTO] { COPY
 WAIT
 ECHO
 RECORDING
 DTG
 DATE
 GRAPHICS
 CLEAR
 MESSAGES } { ON
 OFF } .

(2) SET [AUTO] WAIT [TO] <number> { SECONDS
 MINUTES } .

(3) SET SIZE [TO] size option .

(4) SET FONT { OFF
 ON } .

The FONT option is used on the WIS Workstation only. It allows the user to use the WIS Workstation resident font characters instead of the GIPSY resident characters.

There are many operating environment attributes controlled by the SET command. Generally they will be discussed in the section where those attributes are defined. This section discusses only those attributes which apply to the entire language set.

The words AUTO and TO are included in this statement for readability and clarity of intent. They do not affect the setting of attributes.

The next statement after a SET statement will be performed under the conditions established under the SET statement. All subsequent statements will be affected by the SET.

The COPY function controls the automatic creation of a hard copy via some "video" copier attached to the terminal. If COPY is set to ON, a copy will be made of each display before GIPSY clears the screen to move on to the next function. If COPY is set to OFF, no copies are made -- this is the default mode. This is a Tektronix command and will only work on Tektronix terminals.

The WAIT function controls the timing wait between displays. If WAIT is set to ON, GIPSY will wait for an input from the user (e.g., carriage return) before resuming with the next predefined function. If WAIT is set to OFF, GIPSY proceeds to the next step without waiting for user response. WAIT may also be set to a specific time period before continuing with the next step.

If one sets AUTO WAIT to OFF and AUTO COPY to ON, GIPSY can execute as fast as TSS will allow them to be generated. The effect is to give one a "graphic printer". These two options used in conjunction with a PCS file will allow automatic hard copies of reports from a command sequence using, perhaps, different data for each run.

The ECHO function dictates whether statements processed from a PCS file are to be displayed on the screen as they are read. If ECHO is set to OFF, the outputs are produced but commands from the PCS file are not displayed unless an error is detected. The default mode is ECHO ON.

The SIZE function defines the default character size. This character size will be used for character output not explicitly specified elsewhere. The possible values of <size option> above are the same as the size option defined under the CLASS statement in section 3.2.3.1. If this command is issued from a device which does not have the character sizes defined, the statement will be validated, but will not cause any action. As the character size is changed, GIPSY automatically readjusts all subsequent displays to the new character size.

RECORDING is a special case. It is used to establish whether or not GIPSY will record the statements onto the temporary PCS. RECORDING is ON unless otherwise set. This makes it possible to save all statements entered so they can be recalled at a later time as a syntactically correct PCS file. How this is accomplished is discussed under SAVE/RESTORE.

If DTG (Date Time Group) or DATE options are set on, GIPSY will display the current time in the upper right corner of all reports. The time is obtained from the H6000 computer, and is in the form "24 Jan 90" if date is specified or "1342 24 Jan 90" if DTG is used.

The GRAPHICS option allows the user to turn GIPSY's graphic capability on or off. Normally the setting of this capability is done automatically during GIPSY log-on depending on the terminal type (e.g. graphics is set off on a VIP). However, there may be times when the user has need to control this feature.

The CLEAR option allows the user to override GIPSY's automatic screen clear. Setting AUTO CLEAR to OFF enables the user to place one report or listing on top of another. The default value for this option is ON.

The MESSAGE option allows the user to suppress warning messages and information messages. This capability is especially useful when creating user-friendly interfaces where the appearance of these messages would be distracting. Error messages, which require a corrective action by the user, will still be displayed. The default for this option is ON.

3.2.3.5.1 Define Terminal Command. The DEFINE TERMINAL command allows the user to either alter some of the terminal default characteristics of a terminal or to entirely replace the system-defined parameters of a given terminal with the parameters of another. The syntax for this is:

DEFINE TERMINAL	{	HEIGHT	[-]	<decimal value>	}		
		WIDTH	[-]	<decimal value>			
		SCALE	[-]	<decimal value>			
		PROMPT	[-]	<integer value>			
		CAPABILITY	[-]	<octal value>			
		TYPE	[-]	{		<device number>	}
				<device acronym>			

The terminal HEIGHT and WIDTH refer to the viewable area of the terminal screen in inches. The SCALE represents the number of addressable pixels across the terminal screen. Because these three parameters are interrelated in calculations involving displays, they should be altered proportionately to avoid unpredicted results. The lower left corner of the terminal screen will always be point of origin no matter how these parameters are adjusted.

The prompt character that indicates GIPSY is awaiting input is the symbol ">". This can be changed to another character by entering an octal string of 12 digits after the keyword PROMPT. In order to have the prompt character displayed at the beginning of a new line, it is necessary to input 015012 (carriage return and line feed) as the first six digits of the octal string followed by six more octal digits to define the new prompt character. For example, the command "DEFINE TERMINAL PROMPT 01502077000." will cause the symbol "?" to be the GIPSY prompt character.

The CAPABILITY setting is an octal string which indicates the on/off position of 36 bit switches that represents the terminal-dependent capabilities used in GIPSY processing. Since the resetting of the capability bit cannot overcome the inherent limitations of the hardware of a given terminal, alteration is of limited usefulness beyond the system programming level.

The above parameters are initialized with default values when a GIPSY session is initiated. The user may examine these values by entering the interrupt command //TRMDEP. (Note: the SCALE parameter is stored in variable TSCALE in the TRMDEP area).

The keyword TYPE signals a requested change of all the default characteristics of a terminal to those of another terminal type within the hardware limitations of the user's terminal. The ability to redefine the terminal type would be useful in several instances such as defining a terminal that was incorrectly defined to GIPSY or switching between a VIP (BIS7705) and a WWS (Tektronix/GIPSYmate) modes on the WIS Workstation. The DEFINE TERMINAL TYPE command also makes it possible to create a DAFC containing graphic commands on a non-graphic terminal such as a VIP that has been redefined as one of the terminals with graphic capability and to later produce displays using that DAFC on that previously specified type of graphic terminal.

It should be noted that a redefinition of the terminal type will cause a re-initialization of the internal storage areas related to terminal characteristics and capabilities including any of the other DEFINE TERMINAL commands as well as re-initialization and previously entered SET commands including those involving protections, picture processing, copy, replace mode, etc.

The terminal type can be specified either by the device numbers or the acronyms listed below:

<u>DEVICE #</u>	<u>ACRONYM</u>	<u>TERMINAL DESCRIPTION</u>
01	T4014	Basic Tektronix 4014-1
02	VIP	VIP 7705
03	VIP786	VIP 786W
08	T4014I	Tektronix 4014-1 with Intelligence
09	T4027	Tektronix 4027
10	BATCH	Batch GIPSY
11	T4014W	Tektronix 4014-1 over WIN
13	T4107	Tektronix 4107
18	WWS	WIS/CUC Workstation
20	AT	PC/AT Compatible Workstation
21	MAGIC	WIS Workstation (UNIX)

3.2.3.6 Saving GIPSY Data Structures. GIPSY creates several information structures in the process of graphic query. We call these independent fragments of information subsets. The SAVE statement is the vehicle through which the user can save these subsets for later recall or other uses. The SAVE statement creates a permanent file and writes the designated subset to it. RESAVE will perform the same function except that it will overwrite an existing file rather than create a new one. The action requested by these statements will be taken immediately. The requested subsets will be saved as they exist at the time the SAVE/RESAVE statement is issued. The GIPSY SAVE/RESAVE is slightly different from the standard TSS SAVE/RESAVE in two ways. First, there are several GIPSY subsets to which the GIPSY SAVE/RESAVE can apply, so the subset which is to be saved must be specified. The TSS SAVE/RESAVE only applies to the current file (*SRC). Second, GIPSY removes the file from the Available File Table (AFT) after the information is saved. The last difference is an important one for those users accustomed to TSS. Files that will be used in Batch GIPSY must be created prior to GIPSY execution and saved with a "RESAVE" command.

3.2.3.6.1 Saving the File Descriptor Table (FDT). The command to save or resave the File Descriptor Table discussed in section 3.2.5 is as follows:

```
{ SAVE }
{ RESAVE } FDT [ON] <cat/file string> .
```

The saved FDT file contains the data structure describing a user's data file to GIPSY.

3.2.3.6.2 Saving the Process Control Statements (PCS). All valid statements processed during a user session in GIPSY are written to a temporary file known as the Process Control Statement (PCS) file. At any time, the file may be saved or resaved by issuing the following command:

```
{ SAVE  
  RESAVE } PCS [ON] <cat/file string> .
```

When the SAVE PCS command is processed, the contents of the temporary PCS (i.e., syntactically correct statements that have been processed) are copied to the indicated file. The file is written in standard TSS ASCII format without line numbers. Subsequently, this file can be used as GIPSY input to repeat the same user session at a later time. This file may be edited to create a new PCS and then passed through GIPSY.

3.2.3.6.3 Saving the Graphic Data Set (GDS)/Report. The Graphic Data Set (GDS) is the basis for all statistical reports (Bar Graphs, Pie charts, Gantt Charts, etc.) in GIPSY's Display Module. The GDS is the end result of data retrieval and data manipulation within the Build Tabular Report structure. The GDS may contain any number of reports. Each report comes complete with its own classification, title, and matrix vectors.

By default, the GDS is a temporary system file. The user may save this file by issuing a declarative GDS statement prior to the RUN command or by specifying a SAVE GDS command after the RUN command.

The syntax of the GDS statement is:

```
GDS <cat/file string>
```

The specified file is attached by GIPSY and written when the GDS is created. The GDS statement is a declarative, specifying the file to which all GDS action is directed. Care must be exercised when using a previously saved GDS. If a user-specified GDS is attached at report generation time, the new GDS will be written to the file attached as a GDS regardless of when it was attached.

```
ACCESS GDS <cat/file string>
```

Any time after a graphic data set has been generated, the GDS may be saved or resaved to a permanent file with the command:

```
{ SAVE  
  RESAVE } GDS [ON] <cat/file string> .
```

3.2.3.6.4 Saving the Qualified Data File (QDF). The Qualified Data File (QDF) is the data used as input for the report generation following data selection. The QDF contains only those fields referenced by the user for use in the display, geographic and/or output processor modules. During data selection,

each referenced field for each record which passes the retrieval criteria, is moved to the QDF in the order referenced. Specific order of reference may be forced by use of an INCLUDE statement prior to using any field in another GIPSY statement (see section 3.2.3.10).

By default, the QDF is a temporary system file. If desired, the user may declare the QDF to be a permanent file in the user space by issuing the declarative QDF statement. The syntax for the QDF statement is:

QDF <cat/file string>

The specified file is attached and the QDF data is written to and/or read from this file. The QDF is created in the data selection step and used as input in all other cases. The QDF is discussed further in Appendix E.

The temporary QDF file may be saved at any time after the RUN has been executed by issuing the command:

{ SAVE
RESAVE } QDF [ON] <cat/file string> .

3.2.3.6.5 Saving the Qualified Descriptor Table (QDT). The Qualified Descriptor Table is similar to the FDT except the FDT describes the user's data file while the QDT describes the QDF. The command to save the QDT is as follows:

{ SAVE
RESAVE } QDT [ON] <cat/file string> .

3.2.3.6.6 Saving the Directive Action and File Control (DAFC) File. The DAFC is another temporary file automatically create by GIPSY. The DAFC is a continuously updated control set that reflects the current status of every action or declarative made, including internally-controlled parameters. All the modules of GIPSY communicate through the DAFC. The DAFC is always a temporary file. However, it may be saved into permanent user space with a save command. When the DAFC is saved, a checkpoint of the entire GIPSY system is recorded on the specified file. Subsequently, this DAFC file can be restored to resume execution at the point when the save was taken by issuing a DAFC statement. A previously save DAFC may be restored by issuing this statement:

DAFC <cat/file string>

This statement will cause the specified file to be copied back to the system DAFC, thus restoring the status of every element of GIPSY back to what it was at the time the DAFC was saved. Execution then resumes with the restored DAFC contents and any remaining PCS statements are processed. All required permanent files for which the <cat/file string> was specified will be automatically reattached. However, note that any temporary files in use when the DAFC was saved are not automatically saved and so cannot be automatically

restored. The user must assume responsibility for availability of any data structure normally saved on temporary files (i.e., QDF, GDS, or user file) which must be restored to achieve the desired objective. Furthermore, any DAFC created under a release of GIPSY other than the current version must be recreated to insure full operational reliability. A DAFC may be saved or resaved in one of three ways:

CASE 1

```
{ SAVE  
  RESAVE } DAFC [ON] <cat/file string> .
```

This command will cause two actions to occur. First, GIPSY will take a "snapshot" of itself as it is currently configured. Everything about the current session will be captured. For example, terminal attributes such as background and foreground colors and character sizes, file descriptors, Symbol Tables, Math Tables, Logic Tables, Field Tables, etc. and pointers to permanent files. Secondly, any remaining PCS statements will be executed and remembered. When the DAFC is called into a new GIPSY session, the "snapshot" will be restored. GIPSY will be loaded into the module in which the SAVE command was issued and all executable PCS statements will be processed.

CASE 2

Optionally, you may SAVE or RESAVE the DAFC to resume execution within a specified GIPSY module. The optional syntax is:

```
{ SAVE  
  RESAVE } DAFC FOR <module> [ON] <cat/file string> .
```

where <module> is any valid GIPSY module as specified in section 3.2.2, GIPSY Command Options.

If the "FOR <module>" option is specified, GIPSY will transfer to the specified module. The DAFC will be saved and any remaining statements on the PCS will be executed. When this DAFC is called in to initialize a new GIPSY session, execution will begin in the module specified in the "FOR <module>" option and any unprocessed PCS statements will be processed. All PCS statements following this DAFC save command must be valid for the module specified.

CASE 3

```
{ SAVE  
  RESAVE } DAFC WITH STOP PCS [ON] <cat/file string> .
```

or

```
{ SAVE } DAFC WITH STOP PCS FOR <module> [ON] <cat/file string> .
{ RESAVE }
```

When the option "WITH STOP PCS" is specified any remaining PCS statements will not be executed. When this DAFC is reloaded, GIPSY will load at the appropriate module and begin execution with the PCS statements that had been specified when the DAFC was originally saved/resaved.

3.2.3.6.7 Saving the QDF or GDS in the Data Interface Format (DIF). This option will save the contents of the QDF or GDS to a file as discussed previously. However, the data will be rearranged into a DIF format. DIF is a recognized standard for passing data from one software package to another. The command to create a DIF file is:

```
{ SAVE } QDF
{ RESAVE } DIF FOR [ON] <cat/file string> .
GDS
```

The user must specify the source of the data to be converted to DIF format. This is done by specifying QDF or GDS. If a QUALIFY command is in effect, only those QDF records meeting the qualified criteria will be written to the DIF file. If a LIMIT <vector> command is in effect, only the active vectors of the GDS will be written to the DIF file. Saving the QDF or GDS in DIF format will require more file space than saving files in QDF or GDS format.

3.2.3.7 Saving Data Fields. Occasions may arise when it is desirable to include specific data elements in the QDF which are not referenced by other GIPSY statements. One of the most frequent uses is for including data fields to be displayed when a geographic list will be produced displaying data fields not otherwise referenced. Another frequent use is to force a specific order of data in the QDF.

The syntax of the statement to include specific fields is:

```
INCLUDE { FIELD[S] <fieldname> [<fieldname2> ,...] }
        { ALL FIELDS }
```

This statement simply flags the specified fields as having been referenced and causes them to be included in the QDF and QDT. The INCLUDE statement is defined only in the input specifications. Recall that GIPSY releases the data file once it is read; therefore, fields not referenced prior to the RUN may not be referenced in any geographic display.

The use of a field name in BUILD TABULAR REPORT, SYMBOL TABLE, MATH TABLE, LOGIC TABLE block structures or in the RETRIEVE statement causes the referenced field to be automatically included. Therefore, the INCLUDE statement is only needed if there is a need for fields which are needed but not referenced prior to the run or there is a need for a forced order in the QDF. The automatic inclusion of fields referenced in the RETRIEVE statement can be overridden by the command:

SET [AUTO] INCLUDE OFF.

3.2.3.8 Interrupt Statements. GIPSY has commands by which the user may request information about what has been specified, or reset some environmental parameters without altering the current process. Unlike other GIPSY statements and commands, these commands may be inserted after any prompt character -- even in the middle of another command. For example, you are specifying a retrieval condition, and halfway through you are unable to recall the spelling of a particular field name. Wouldn't it be nice to be able to list your file description to check the spelling? The interrupt statement provides that type of support.

Every interrupt command begins with a double slash (//), to let GIPSY know you are interrupting, immediately followed by the command. This type of command cannot end with a period. The // must be in the first two positions of the input line. If an interrupt command is not recognized, it is ignored and is considered to be a comment. If interrupt commands are included in the PCS file, the command is executed but is not echoed back to the terminal. Interrupt commands are not recorded on the temporary PCS and thus are not preserved by the SAVE/RESAVE command. The following interrupt commands are defined as shown.

//CAT [<literal>] - same as //CATEGORY.

//CATEGORY [<literal>] - displays the name of all categories. If the <literal> is specified, the display will include only those names starting off with the <literal> .

//CIRCLES - causes GIPSY to display the names of all geographic circle sets you currently have defined to GIPSY.

//CLEAR PAGE - clears the screen to start a new page; if the automatic copy feature is on, a copy will be taken prior to clearing the screen.

//COLS [<literal>] - displays the name of all columns. If the <literal> is specified, the display will be restricted to those names beginning with the specified <literal> .

//CONTINUE - causes GIPSY to assume that the current report or graphic page is complete. GIPSY will then continue with the next command.

//COPY - automatically copies the contents of the current display regardless of the automatic copy parameter.

//COPY OFF - turns GIPSY's automatic hard-copy feature off.

//COPY ON - turns GIPSY's automatic hard-copy feature on.

//ECHO HOLD - command will save the current status, either ON or OFF, of the echo process.

//ECHO OFF - terminates the echo of PCS supplied statements back to the terminal. Only statements containing errors are displayed on the terminal.

//ECHO ON - causes all noninterrupt statements from a PCS file to be echoed back to the terminal.

//ECHO RESTORE - will reset the echo status back to its previous state, either ON or OFF.

//FDT - causes GIPSY to display the field names in the current File Descriptor Table.

//FDT [<literal>] - causes GIPSY to display the current File Descriptor Table entries beginning with the same characters as the <literal> specified.

//FDT [(ONLY)] - Displays only field names defined as FILE or FILE Extended fields.

//GDT - displays field names defined as GLOBAL fields.

//HUH - causes GIPSY to repeat prompted error message.

//IDSII STRUCTURE - displays the File Structure Table (FST) hierarchical record structure including the record types of the current I-D-S/II integrated data file and all valid index key names.

//IDT - allows the user to get a list of all field names associated with the index file process.

//LAST - displays last input statement that is incomplete or not yet acted upon.

//LIMIT vector mode - displays the name of all vectors within the current vector mode limit definition. The vector modes are ROW, COLUMN, CATEGORY, and SECTION.

//LIMIT CIRCLES - displays the name of all the circles currently included in a limit defined for circles.

//LIMIT SYMBOLS - displays the name of all the symbol sets currently included in a limit defined for symbols.

//LIMIT TRACKS - displays the name of all the track sets currently included in a limit defined for tracks.

//LIST [n] lists the last <n> statements entered during the current session (i.e., content of temporary PCS). If <n> is not specified, all statements will be listed.

//LIST <literal> [<end literal>] - list all statements beginning with the first occurrence of <literal> to the first occurrence of <end literal>. If no <end literal> is specified all statements containing <literal> will listed.

THIS PAGE INTENTIONALLY LEFT BLANK

//LOCATION - lists the location name and associated coordinate position of geographic locations defined in a LOCATION TABLE block structure.

//LOGIC - displays the names defined in a LOGIC table.

//MATH - displays the names defined in a MATH table.

//NOTE - causes the text following NOTE to be displayed on the terminal as a message to the user of the PCS; text may be up to 126 characters on a single line. A blank space must separate the keyword NOTE from text.

//PAUSE [n] - causes GIPSY to pause for <n> seconds before continuing with additional activities.

//PICTURE - lists the names of pictures stored on the picture file.

//PREVIEW [n] - allows user to preview unprocessed PCS statements by listing statements in the current PCS which have not yet been processed. If <n> is not specified, all statements will be listed.

//PROCESS - lists the names of the GIPSY processes which have been built through the use of the DEFINE PROCESS command.

//PURGE - purges all remaining statements from the current PCS input file.

//PURGE [n] - purges the next <n> statements from the current PCS input file.

//PURGE GROUP - purges the last group of statements supplied by a PCS statement. This command would be used to purge a group of inputs which were found to have too many errors to bother attempting to fix.

//QDT - Displays all fields defined as QDF, QDF extended, GLOBAL fields, QUALIFY fields and all FILE and FILE extended fields which have a QDT starting position.

//QDT [<literal>] - causes GIPSY to display all fieldnames in the QDT. If a <literal> is specified, the display is restricted to those fieldnames beginning with the same characters as the <literal> specified.

//QDT [(ONLY)] - Displays only field names defined as QDF or QDF extended fields and all FILE and FILE Extended fields which have a QDT starting position.

//QLT - Displays field names defined as QUALIFY fields.

//REPORT - causes GIPSY to display the report identifier during statistical display processing.

//RETURN - causes GIPSY to return to the previous input mode, i.e., if reading from a PCS return to terminal mode; if in terminal mode return to PCS mode.

If there is no PCS or if all statements have been processed from the PCS file, the statement will be ignored.

//ROUTINES - provides a list of the user subroutines currently loaded.

//ROWS [<literal>] - displays the name of all rows. If the <literal> is specified, the display will be restricted to those names beginning with the specified <literal>.

//SEC [<literal>] - same as //SECTION.

//SECTIONS [<literal>] - displays the name of all sections. If the <literal> is specified, the display will include only those names beginning with the specified <literal> .

//SYMBOLS - causes GIPSY to display the names of all geographic symbol sets you currently have defined to GIPSY.

//TEXT - Causes GIPSY to display the names of all text entries in the text table.

//TEXT [<literal>] - Causes GIPSY to display text table entries beginning with the same character as the literal specified.

//TEXT NAMED <text name> - Causes GIPSY to display all lines of text associated with the specified <text name> .

//TRACKS - causes GIPSY to display the names of all geographic track sets you currently have defined to GIPSY.

//WINDOW - lists the window name and coordinate description of geographic windows defined in the WINDOW TABLE block structure.

3.2.3.9 Comments. Any line beginning with a double slash (//) followed by at least one blank is considered a comment. Comments are not displayed at the terminal (from a PCS); they are not recorded onto the temporary PCS and have no effect on GIPSY processing.

3.2.3.10 Executing Other TSS Commands. The experienced TSS user will occasionally find it useful to get back to TSS to perform some function and then to return to GIPSY at the point of departure. GIPSY's TSS command provides this capability. The TSS command consists simply of the string

TSS [<single TSS command>]

After this command is recognized, GIPSY will interpret all subsequent inputs as commands to be passed to the GCOS Time-Sharing System (TSS). This state is indicated by the prompt character "<". A null input will return you to normal GIPSY operations. Some potential applications of this function are to create or access files, to edit saved SY PCS files, to execute a user module, etc.

If a single TSS command is included on the TSS statement, GIPSY will pass that command to TSS and automatically return to normal GIPSY operation.

3.2.3.11 Clearing Data Structures. The CLEAR statement provides the capability to clear classification and title settings and to clear all previously entered statements from the temporary PCS. The syntax of the statement is:

```
CLEAR { CLASS
      TITLE
      PCS } .
```

3.2.3.12 Executing a User Program. User generated executable modules (H*) may be executed within the GIPSY syntax processing. Supplemental user functions may be interjected by instructing GIPSY to execute a TSS compatible H* by issuing the EXECUTE command. This command has the syntax:

```
EXECUTE <cat/file string of user H*>.
```

This command will cause GIPSY to swap itself out and turn control over to the executable module defined by cat/file string of user H*. Upon completion of the user program, GIPSY is restored to the state it had immediately prior to the issuance of the execute command.

3.2.3.13 Terminating the GIPSY Session. The user can exit GIPSY by entering the statement

```
DONE.
```

The DONE command can be entered at any point where GIPSY is not expecting specific input. If the user is being prompted for an error correction, the error must first be corrected or deleted. If the user is in the middle of a block structure, it must first be completed with an END statement before the DONE will be accepted. All files, which GIPSY has attached, will be released upon the termination of the GIPSY session.

3.2.3.14 GIPSY Color Processing. GIPSY provides the user with the capability of producing graphic reports in multiple colors. Available color names and definitions are contained in the terminal definition file (..TRMDEF). The number of colors which can be displayed at one time is determined by the particular graphics terminal (7 for the Tektronix 4027, 8 for the WIS Workstation). GIPSY gives the user the capability of setting the color in which maps and graphs will be drawn as well as setting the background color through use of the SET COLOR statement. The exception to this is the WIS Workstation which cannot modify background color. The format for the SET COLOR statement is:

```
SET { COLOR
    BACKGROUND COLOR
    BACK } <color name> .
```

3.2.3.15 User Defined Subroutines. GIPSY has the capability of passing data between itself and a user-built subroutine. Such a subroutine would be used to provide a data manipulation capability which GIPSY does not have. As an example, GIPSY could pass two coordinate values to a subroutine which would calculate the distance between the two points and return that value to GIPSY. A user's subroutine library is accessed by the command:

LIBRARY <cat/file string of library>.

The LIBRARY statement identifies the file containing the user subroutines if the cat/file string is supplied. If the statement is terminated without supplying a file, the system-supplied user-subroutine library is accessed. One LIBRARY statement must be supplied for each library to be searched. User-supplied subroutines take precedence over system-supplied subroutines. If the user library has duplicate subroutines, the last subroutine loaded will be used. (A LIBRARY statement is not required to access the GIPSY-supplied \$DATE, \$TIME, or \$ISPPTR subroutines).

3.2.3.16 JDAC Command. GIPSY has the ability through the JDAC command to access a Direct Access Communication (DAC) program such as the WIN File Transfer Service (FTS) subsystem, VIDEO (proper permissions are required), or any other valid DAC program. The JDAC command is available within the GIPSY modules SYNTAX, GEOMOD, GDRMOD, and DISPLA. It is operationally equivalent to the JDAC command within the Time Sharing System (TSS). The JDAC command consists simply of the string:

JDAC <DAC program name or SNUMB>

Once GIPSY recognizes the JDAC command, it is executed and the user will then be able to interface directly with the requested DAC program. Upon termination, some DAC programs return to the invoking subsystem while other DAC programs require user direction. Because of this, control may or may not return to GIPSY. For further information, refer to the termination sequence of the invoked DAC program.

3.2.4 User Data Input. Most graphic systems require that you pre-format your data into some graphic data file. In GIPSY, the data used to create graphic displays typically comes from a user's existing data file. You do not type in sets of x-y coordinates to create graphs. Based on your specification, GIPSY synthesizes the data from your file to produce the graphic details. Your data file can be a large master file, a data subset created for graphic purposes, or simply a card image data file entered via the keyboard, a user program, card deck, etc. In any case, you must identify the file to GIPSY. In most cases, GIPSY will process the entire file sequentially. GIPSY has an extremely fast input/output package so this usually takes only a few seconds. Data retrieval can become time consuming for a large, complex file (in excess of 4 million characters). For large UFAS sequential, relative, or indexed files, I-D-S/II indexed files, and GFRC ISP files, GIPSY provides an index capability which allows GIPSY to avoid reading unneeded portions of the file, resulting in significant savings in execution time.

3.2.4.1 The Data File. Your data file is identified to GIPSY with the FILE statement which has the following syntax:

FILE $\left\{ \begin{array}{c} \text{<cat/file string>} \\ * \end{array} \right\}$

The <cat/file string> is the H6000 catalog file string locating the data file. It is specified using the standard conventions of file access.

For GFRC and I-D-S/II integrated files GIPSY will remove the file from the Available File Table (AFT) if the file was not already in the AFT (i.e., full cat/file used). If GIPSY found the file in the AFT, it will leave it in the AFT. The asterisk notifies GIPSY to use the TSS current file for the data file.

For a UFAS sequential, relative, or indexed file or an I-D-S/II indexed file, the file must be specified using the full cat/file string in the FILE statement. If the file already exists in the AFT, a new reference will be loaded into the AFT. As soon as GIPSY's data selection process is complete, GIPSY releases its access to the user file.

A data file may be either TSS ASCII or BCD. GIPSY will translate the ASCII data to BCD for GIPSY processing for most file types.

3.2.4.2 The Index File. GIPSY can retrieve data using file indexes. For UFAS indexed files and the I-D-S/II indexed and integrated files, GIPSY uses the indexing facilities provided by these file formats. The user must know the locations of the index fields (also known as key fields) within the record of the user's data file. The index fields must have been defined as either prime or alternate keys when the data file was created. For an I-D-S/II integrated file the valid key fields will be displayed at the end of the GET IDSII STRUCTURE statement, the //IDSII and //FDT statements. When the GIPSY user builds the File Descriptor Table (FDT) within GIPSY (see section 3.2.5.1), the index fields to be used must be defined as they occur in the data record of the user's file. The field names can be different, but the starting positions of the index fields must match those defined in the user's file. There is no need to further describe the index fields for UFAS indexed or I-D-S/II files, that is, do not use the ADD TO INDEX statement in the FDT block structure. When using the INDEX statement for UFAS indexed or I-D-S/II files, the index file must be specified using the full cat/file string.

For ISP files, the GIPSY user can build a GIPSY index file. The GIPSY index file is a file to identify the location of key data elements within a large GFRC ISP data file. Building the GIPSY index file is discussed in section 3.2.5.2. This file is not to be confused with the ISP index. GIPSY's index file is a small sequential file consisting of major key data elements and an associated ISP pointer to the data record. GIPSY first reads the index file looking for conditions specified by the user. When a condition is satisfied by

looking at the index, GIPSY then reads only the corresponding record from the data file.

Consider the illustrated ISP data file and its associated index in figure 3-3. Let us assume an example in which the user needs only the data records in which the ULC field contains the code "BN" and the DE field contains the code "Q". GIPSY would look at the index and find that only units 2 and 7 need to be read from the data file. GIPSY then uses the \$ISPTR values to go directly to the data records in the ISP data file containing units 2 and 7. The remainder of the file will not be read.

The syntax of the statement identifying the index file is:

```
INDEX      <cat/file string>
```

The index file may be created by a user application program or it may be created by GIPSY as a data subset.

3.2.5 Describing the Data Records. The file records which the GIPSY user must describe are the data file records and the index file records (if an index file is used). The data in the index file are generally the same as selected fields from the data file. However, these are two separate fields for separate purposes. Consequently, the fields described for the data file are separate and distinct from those in the index file, when the user has an index file.

3.2.5.1 Describing an I-D-S/II Integrated file. The command GET IDSII STRUCTURE must be entered after an I-D-S/II integrated file is designated as the current file in the FILE and INDEX statements. This command results in the display of the record type identifiers to be used in subsequent GIPSY commands to identify a specific record type. This command also causes the creation of the File Structure Table (FST) which defines the I-D-S/II structure that will be stored with the File Descriptor Table (FDT) described in subparagraph 3.2.5.2.

3.2.5.2 Describing Data File Records to GIPSY. GIPSY allows the user to describe his/her data file to GIPSY rather than forcing a predetermined format on the user's data. The data records are described to GIPSY by means of a File Descriptor Table (FDT). The FDT may be recalled from a previously saved file containing an FDT or it may be specified in-line. In any case, the data fields must be defined prior to attempting to use them in GIPSY statements.

If an acceptable FDT already exists, the data fields described therein may be made available to GIPSY by issuing the command

```
FDT      { <cat-file string>
          * }
```

An * indicates that the FDT is in the TSS current file. The catalog file string points to an ASCII or BCD file containing a GIPSY File Descriptor

Table. This file could have been built and saved through GIPSY as described below or it could have been produced by some other means.

If the data file is an I-D-S/II integrated file, a File Structure Table (FST) that defines the hierarchical record type identifiers of the file will have been stored with the FDT and will also be made available when the FDT command described above is issued.

If a File Descriptor Table does not exist, one may be specified in line by describing the data records in the block structure bounded by BUILD FDT and END. This block structure may also be used to modify an existing FDT. The syntax for the structure is:

THIS PAGE INTENTIONALLY LEFT BLANK

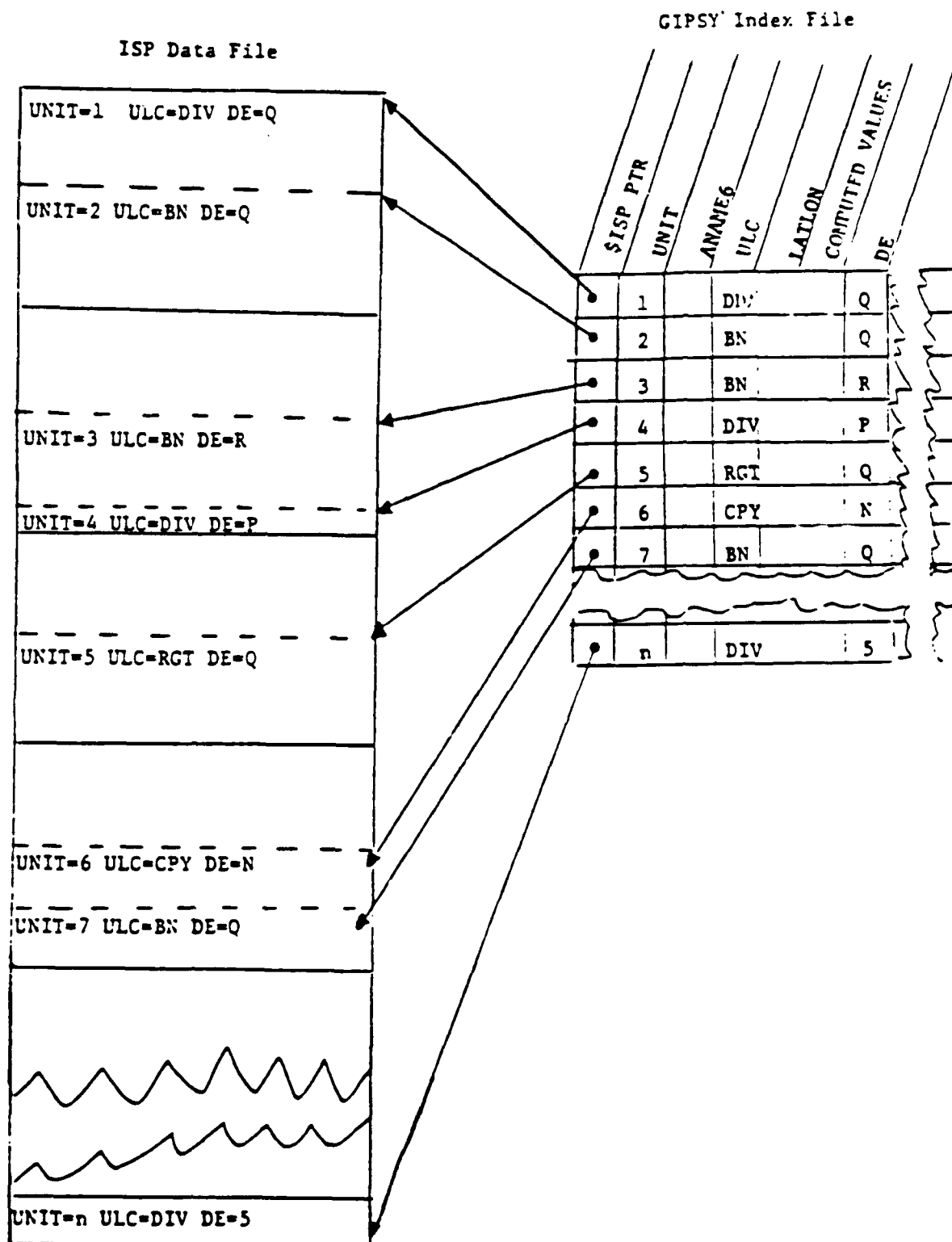


Figure 3-3. Index/File Relationship

BUILD FDT.

[<commands to ADD field definitions>].
[<commands to INSERT new field definitions>].
[<commands to DELETE old field definitions>].
[<commands to LIST the FDT>].
[<commands to SORT the FDT>].

END.

The <commands to ADD field definitions> consists of the key words ADD TO along with one of the options listed below and as many definitions as needed to create a new file description or to augment an existing file description.

ADD TO { FILE
 INDEX
 GLOBAL
 QUALIFY
 GENERIC
 <record_type> }

<fieldname> { <defined fieldname>, [<type and length>] }
 <starting position>, <type and length> }

<fieldname2> { <defined fieldname>, [<type and length>] }
 <starting position>, <type and length> }

.
.
.

<null line>.

The ADD TO FILE clause is the default mode for the BUILD FDT structure. It indicates that the FDT definitions apply to the entire specified data file. The single word "ADD." following the BUILD FDT command has the same result as the ADD TO FILE clause. This clause is not valid for I-D-S/II integrated data files.

The ADD TO INDEX clause is discussed in subparagraph 3.2.5.3. The ADD TO GLOBAL and ADD TO QUALIFY clauses are described in subparagraph 3.2.5.4.

The ADD TO GENERIC option will allow the creation of a generic record level FDT entry to be applied against record types that are not covered by the more specific ADD TO <record_type> entry. This could be used to retrieve data from some types of records as a block of data rather than individual fields. This clause is only valid for I-D-S/II integrated data files.

The <record_type> is a unique four digit identifier assigned to each record type in the I-D-S/II integrated hierarchical record structure. The ADD TO <record_type> will allow the creation of a record level FDT definition for a specific record type. An ADD TO <record_type> clause will have to be created within the FDT for each record type to be accessed. The field names for each record type must be unique within the entire FDT. The ADD TO <record_type> option is only valid for I-D-S/II integrated data files. For I-D-S/II integrated data files any field names that are packed decimal must be described in the FDT up to the point where no further fields are described for that <record-type>.

Upon recognizing the FDT ADD clause, GIPSY will display a message indicating that it is ready to accept a field definition. This message will be repeated after each field is successfully defined. Then the next new field definition may be entered. A <null line> (e.g., carriage return) will return you to the mode to enter another FDT command. The <fieldname> is a user-specified name that will be used to reference the data starting in the character position within the record specified by starting position. The name may have a length of 1 to 12 alphanumeric characters, the first characters of which must be alphabetic. The name may contain imbedded hyphens or underscores. No other special characters are permitted. The <starting position> must be an integer value defining the location within the data record where GIPSY is to look for the data. Alternately, it may be a previously <defined fieldname>. This option will be discussed later in this section. The length and type of data contained in the defined field is specified by <type and length>. The <type and length> must be specified in one of the following forms:

A <n> The type indicator A indicates alphanumeric data. The length indicator n must be specified as an integer value designating the number of characters in the field. There must be no space between the type and length (e.g., A132 indicates a 132 character data field containing alphanumeric data). The A can be used for numeric data; GIPSY will treat it as an integer in computations and as alphanumeric at all other times. Partial field definitions can be applied to this type of data.

THIS PAGE INTENTIONALLY LEFT BLANK

I <n> The type indicator I designates integer data. Integer type data consists of a string of numeric digits (0-9). The length of the integer field is specified as an integer value n immediately adjacent to the type indicator I (e.g., 16 represents an integer field with six digits). Integer data is assumed to be right justified in the field; blanks will be treated as zero. If a record contains an alphanumeric value in a type I field, the value will be treated as null.

C <n> Type C indicates coordinate-type data. Specifically this is composed of a latitude and longitude in one of the following formats:

<u>Format</u>	<u>Length (n)</u>
DDHDDDH	7
DDMMHDDDM:1H	11
DDMMSSHDDMMSSH	15

where the D's represent degrees, M represents minutes, and S represents seconds. That portion which precedes the first H is latitude, and that H must specify the North-South hemisphere - N for North, S for South. The part which follows the first H represents longitude and the last n must specify the East-West hemisphere - E for East, W for West. The minutes and seconds may contain integer values from 00 through 59, inclusive. The latitude is one to three sets of integers whose value ranges from 00 through 90 degrees; the longitude ranges from 000 through 180 degrees. GIPSY will ignore any invalid coordinate.

F <w.d> Type F is used when the field contains floating point (or real) data values. The decimal point may be physically in the data field or it may be implied by the description of the field. The w indicates the width (or length) of the field and the d indicates the number of digits to the right of the decimal point contained in the field. A decimal point appearing in the data will override the number of decimal places specified in the field definition. For example, assume that a data field TALLY is defined as F7.3, a value of 1234567 in TALLY will be read by GIPSY as 1234.567 while a value of 1.23456 will be read as 1.23456.

P <w.d> The type P designates a packed decimal value. The packed decimal type is only valid for the I-D-S/II integrated file. It is stored as two digits per 9-bit byte. When read, the packed decimal is expanded into GIPSY's internal format. The packed decimal value will be treated as either integer type data or floating point data, depending upon the number of digits to the right of the decimal point. The w indicates the width (or length) of the packed decimal field. The width of the field must include the space required for the sign if present and the

number of positions right of the assumed decimal point (do not include the decimal point in the width). In short the width is the sum of digits left and right of the decimal point and space for the sign if present. The d indicates the number of digits to the right of the decimal point. The d may be omitted when it is zero.

- BI The type BI designates a binary integer value. The binary integer does not have a user specified length. The field must be a 36-bit binary word in processor memory format. The length is thus assumed to be 6 BCD characters.
- BF The type BF designates a binary floating point value. The BF item represents floating point data in internal H6000 format of 36 bits. A length of one word (6 BCD characters) is assumed and is not user alterable.

THIS PAGE INTENTIONALLY LEFT BLANK

L.<n> The Type L indicates a logical bit field where n indicates the bitnumber within the 6 bit BCD character (the leftmost bit is bit 0). The logical bit field may only be used as a true/false condition within a conditional expression. 0 is false, 1 is true. A definition of:

COMBAT-READY 24 L.4.

indicates that the fifth bit of character 24 in the record is to be treated as a logical variable. This field could be used in a conditional expression such as RETRIEVE IF COMBAT-READY AND SERVICE EQ ARMY. This condition would be satisfied for all ARMY records in which the 5th bit of character 24 is a one.

A null line may be entered in response to the new entry message to terminate the ADD process. Another BUILD FDT command may be entered at this time or an END statement may be entered to terminate the BUILD FDT block structure.

It was mentioned earlier that a previously defined fieldname could be used as the <starting position> instead of an integer number. When this is used as the <starting position> the starting position, length, and type attributes of that field are provided to the new field as defaults. If <type and length> are specified, the specified parameters, of course, override the defaults. Recall that partial field notation can be appended to any field name; this applies here as well. Observe the following sequence:

DATE 26 A9.	may be used to define a field containing data in the format of 15 JAN 80 beginning in position 26 in the data record.
DAY DATE (1/2).	creates a 2-character alphanumeric field starting in position 26 called DAY. Starting position is taken from definition of DATE.
MONTH-YR DATE (4/9).	defines a 6-character alphanumeric field starting in position 29 called MONTH-YR.
YR-DIGIT MONTH-YR (6).	defines a 1-character alphanumeric field starting in position 34 called YR-DIGIT (e.g., the 0 in 15 JAN 80).

This type of field definition is called a relative field definition.

GIPSY provides a mechanism for extending the data record at execution time in order to move data around, perform computations, etc., and to put data in the extended portion of the data record. To accomplish this, there is a special case when an "*" may be used (instead of an integer number) for the <starting position>. This option may be used only to define fields whose data will be loaded by a FIELD TABLE. In this case the "*" is defined as the next double word position after the end of the highest (rightmost) currently defined data

location. For example, if we build on the above definitions of the DATE field with:

```
WORK-AREA * A109.
```

the starting position of work area would be position (column) 37 (i.e., the next multiple of 12 plus 1). If relative field definitions are used, the user need not be concerned with actual start positions. The use of this area will be described in detail in the FIELD TABLE discussion on modifying user file data.

The <commands to insert new field definitions> is in the form:

```
INSERT    { BEFORE }    <old fieldname>.
           { AFTER  }

           { <starting position>, <type and length> }
<field name> { <defined fieldname>, [<type and length>] }
.
.
.
<null line>
```

The INSERT function is identical to the ADD function except that the definitions are inserted before or after a previously entered field definition, as specified.

The commands to delete old field definitions is in the form:

```
DELETE    <fieldname> [THROUGH <fieldname n>].
```

Field definitions can be deleted only by specifically identifying a single fieldname or by specifying a range of fieldnames by adding the THROUGH option. If the THROUGH option is used any definitions appearing in the FDT at the time of the delete may be used to define the limits of the deletion. If a field has already been referenced outside of the FDT block structure, that field definition cannot be deleted. (The column headed "QDT POSITION" on a LIST is nonblank if the field has been referenced).

The <command to list field definitions> is in the form:

```
LIST      [<fieldname>] .
```

If the <fieldname> is not specified, a formatted list of all entries currently defined in the FDT will be displayed. If <fieldname> is specified only the named entry is listed. The interrupt command //FDT and its permutations may be used to augment the reviewing of the FDT.

The commands to sort the FDT provide the capability to rearrange the FDT, based upon either the alphabetical order of the fieldname or the defined starting position for the data.

The syntax for these functions is:

SORT { FIELD }
 { STPOS }

SORT FIELD will rearrange the field definitions such that the fieldnames appear in ascending alphabetical sequence. SORT STPOS will rearrange the field definitions such that they appear according to the ascending value of the starting position.

The BUILD FDT function may be used to create a new FDT or to augment an old one. If an FDT statement has been previously entered, or if an FDT has been defined by a previous BUILD FDT block structure within the current session, subsequent BUILD FDT functions will operate on the defined FDT entries as well as allowing the addition of new ones.

The BUILD FDT block structure must be terminated by an END statement. Once out of the block, the GIPSY SAVE command may be used to save the FDT on disk for later recall. Of course, it may not be necessary to save the FDT since all of the BUILD FDT commands are saved on the PCS file, just as all other GIPSY statements are saved. This effectively allows a permanent in-line FDT. Neither the use nor definition of a field within the BUILD FDT block structure constitutes a field reference to cause it to be included in the QDF.

3.2.5.3 Describing the Index File. The index file is described using the same tools used to describe the data file. The BUILD FDT block structure described in section 3.2.5.2 is used as described except that the ADD statement must be:

ADD TO INDEX.

This feature is only valid for ISP files. All other features and requirements are as described in the preceding section.

The index descriptors may be included with the data file descriptors, or they may be separately specified. Since the index descriptions can have the same fieldname as the file descriptors, the latter is recommended for clarity. The index fieldname can be used only in statements which reference the index. The converse is also true.

The following example shows an index description:

```
>BUILD FDT.  
>ADD TO INDEX.  
ENTER NEW FIELD OR CARRIAGE RETURN TO TERMINATE
```

```

> LOCATION 1 A12.
ENTER NEW FIELD OR CARRIAGE RETURN TO TERMINATE
> MISSION 15 A5.
ENTER NEW FIELD OR CARRIAGE RETURN TO TERMINATE
>END.
>BUILD FDT.
>ADD.
ENTER NEW FIELD OR CARRIAGE RETURN TO TERMINATE
> DATE 25 I6.
ENTER NEW FIELD OR CARRIAGE RETURN TO TERMINATE
> MONTH 27 I2.
ENTER NEW FIELD OR CARRIAGE RETURN TO TERMINATE
>END.
>RETRIEVE FROM INDEX IF LOCATION EQ ENGLAND.
>//FDT

```

FIELD	POSITION	LENGTH	TYPE	DECPT	QDT	POSITION
LOCATION	1	12	A			<INDEX>
MISSION	15	5	A			<INDEX>
DATE	25	6	I			
MONTH	27	2	I			

>

3.2.5.4 Describing GLOBAL and QUALIFY Records. Unlike the FILE AND INDEX records, GLOBAL and QUALIFY records do not describe a FILE. These records represent internal working storage. The ADD statement must be:

```

ADD TO { GLOBAL
        QUALIFY } .

```

The fields are defined as discussed in section 3.2.5.2 except the <starting position> must be an asterisk, "*", or previously defined field name of the same type.

3.2.6 Conditional Expressions. Many GIPSY statements or structures are set up to perform the required functions only when some set of conditions is satisfied.

For example, you might want to retrieve a record only if it is an Army record. The GIPSY language element for stating that criteria is called a conditional expression. Unfortunately, conditional expressions tend to be rather technical and at times programmerish. This is necessary because of the precision of the specifications required. A conditional expression is composed of one or more logical comparisons joined together by the connectors AND and/or OR, and possible parentheses to control the order and grouping of comparisons. Each logical comparison has one of the following forms:

- (1) <fieldname> <relational operator> <value> [, <value>, ...]
(only for EQ, NE, BT)
- (2) <fieldname> <relational operator> & <fieldname>

- (3) <logical field> up to 6 bits in one character (L.0-L.5)
- (4) <fieldname> CHANGES compares present to previous
- (5) <fieldname> COMPLETE compares present to next
- (6) <coord fieldname> WITHIN <coord>, <coord> [, <coord>, <coord>, ...]
- (7) <logic names>
- (8) (<logical comparison> ^{AND}
OR <logical comparison> AND logic ' comparison>)
- (9) <logical comparison> ^{AND}
OR < logical comparison>...
- (10) NOT <logical comparison>

Each of the above must be considered to be a Boolean operation. The condition is satisfied when the answer to the comparison is true; otherwise, it is false.

In each of the cases above, <fieldname> is any valid field reference defined in the currently specified FDT. In case (1) above, multiple comparison values (i.e., [, value ,...]) are permitted only in comparisons for equal, not equal, and between. Consistent with the foregoing limitation, the <relational operator> may be any of the following:

<u>Relational Operator</u>	<u>Meaning</u>
EQ	is equal to
NE	is not equal to
BT	between
LT	is less than
GT	is greater than
LE	is less than or equal to (inclusive)
GE	is greater than or equal to (inclusive)
=	is equal to
<	is less than
>	is greater than

In case (1) the contents of the data in the record identified by <fieldname> will be compared against the literal value using the relational operator to determine whether the comparison is true or false. The literal value may be of any valid data type. It will be converted to the type associated with the <fieldname> for comparison. This literal must be enclosed in quotes if it contains any special characters (a decimal point in a floating point number is not a special character); it may be enclosed in quotes for any alphanumeric comparison, if desired. Any item enclosed in quotes will always be treated as alphanumeric, even if it is a number. If the literal value does not agree in length with an alphanumeric data field, the value will be padded on the right

with blanks if too short, or truncated on the right if too long. The length is unimportant in numeric (I or F type) comparisons.

Case (2) is identical to case (1) except that the parameter on the right of the relational operator will be taken from the data field identified by <fieldname>. The "&" alerts GIPSY that the string of characters identified by the second <fieldname> is a data field reference rather than a literal value.

In case (3) <logical field> refers to any field in the FDT with a type of L. This is a 1-bit data reference that is considered to be a completed logical comparison. A value of true or false is assigned dependent upon the contents of the defined bit. A bit value of zero in the data indicates a false condition; a bit value of one indicates a true condition.

Case (4) compares the current value of the data identified by <fieldname> with its previous value. The condition is true if the current field value is different from the previous field value; otherwise it is false, i.e., the field value does not change. The field changes for the first record by definition.

Case (5) is somewhat similar to case (4). It compares the current value of the data identified by <fieldname> with what its value will be in the next record. The condition is true when the next value of the field will be different from the current one, i.e., a data record grouping is complete because the next one will start a new group. By definition, the record grouping is always complete for the last record. This conditional expression is not valid in the BUILD TABULAR REPORT block structure.

Case (6) effectively performs an area search. Each <coord> must be a geographic coordinate. These coordinates must occur in pairs representing a geographic rectangle. The first coordinate of the pair represents the lower left corner of the geographic rectangle and the second represents the upper right corner. The WITHIN operator causes GIPSY to determine if the data file coordinate defined by the <coord fieldname> is within the rectangular area defined by the coordinate pair. The <coord> pair may be repeated as often as desired to describe an area. If multiple pairs are supplied, GIPSY will repeat the test for each coordinate pair; the condition is true if the data falls inside any of the rectangles. The rectangles may overlap without redundant selection of data. This capability provides a very fast random area search capability. The searching is done by simple comparisons rather than by time consuming complicated mathematical computations.

Case (7) uses a name assigned to a conditional expression in a LOGIC TABLE as the logical comparison. It has a true/false value based upon the evaluation of the conditions defined for that item in the LOGIC TABLE (see section 3.2.7 for details on the LOGIC TABLE).

Cases (8) and (9) define <logical comparisons> recursively. They emphasize that one or more <logical comparisons> can be collected and connected together (using parentheses and connectors) and treated as a single logical comparison

having a true/false value. This definition is deliberately recursive. Any of the preceding cases may be substituted in case (8) for <logical comparison> including case (8) and (9) themselves. Eventually of course, each <logical comparison> has to be replaced by one of the <logical comparisons> in cases (1) through (7). Be sure to take note that even the complex string of <logical comparisons> derived by repeated expansions into case (8) can be treated as a <logical comparison> with a single true/false result.

The last sentence above is particularly important when we consider case (10). Case (10) says that the inverse of that comparison results by appending a unary NOT operator. The NOT operator applies to the <logical comparison>. It cannot be used in front of the <relational operator> or the <value>. It may read a little strange, but remember, the NOT reverses the result of an entire logical comparison or string of comparisons. For example:

MONTH EQ FEB AND DAY EQ 29

will select leap year day, while

NOT (MONTH EQ FEB AND DAY EQ 29)

will select everything except leap year day. However,

NOT MONTH EQ FEB AND DAY EQ 29

will select only the 29th of every month except FEB. Finally,

MONTH NOT EQ FEB AND DAY NOT EQ 29

will result in syntax errors because the NOT operator can modify only the logical comparison, not the relational operator.

When all the pieces of <logical comparisons> are in place, it is called a <conditional expression>. Throughout this manual the term <conditional expression> will refer to that definition provided here.

To avoid ambiguity, let us call two or more logical comparisons connected by AND or OR, a phrase. When compounding logical comparisons into a phrase, care should be taken in establishing the precedence of evaluation. All logical comparisons connected by an AND must be true for the phrase to be true; when any logical comparison connected by OR is true, the phrase is true.

Parentheses should be used to collect OR connected phrases or logical expressions into a single truth value if the OR connected phrase is not all-inclusive. Perhaps an example will help to clarify this. The expression:

DAY-OF-WEEK EQ MONDAY AND DAY LT 10 OR DAY GT 20

will qualify all Mondays in the first 10 days of the month plus all days of the week for dates greater than 20. While:

DAY-OF-WEEK - MONDAY AND (DAY LT 10 OR DAY GT 20)

will only select Mondays before the 10th and after the 20th. What has happened in the latter case is that the DAY phrase was resolved to a single truth value prior to being connected by the preceding AND. To state it another way, AND connects the logical comparisons on each side of it; OR allows the logical expression to stand alone; and, the parentheses allow a series of phrases to be collected so that everything in the parentheses is treated as a single phrase. Parenthetical expressions may be nested within parenthetical expressions to an unlimited depth.

The syntax of the items on the right hand side of the between relational operator (BT) is slightly different from the others. BT accepts a pair of comparands separated by a slash (/). The comparison is inclusive of the values on the end of the range.

If a user wanted to select all data for the 1st through the 5th, the 13th through the 18th, and the 21st through the 30th of December, the conditional expression would be:

MONTH EQ "DEC" AND DAY BT 01/05, 13/18, 21/30

This of course assumes that MONTH and DAY are fields in the FDT currently loaded by GIPSY.

OR is normally assumed for comparands in a list on the right-hand side of a logical comparison. The conditional expression above will be interpreted as:

MONTH EQ DEC AND (DAY BT 01/05 OR DAY BT 13/18 OR DAY BT 21/30)

Take special note of the phrases within parentheses. Note that you cannot say DAY BT 01/05 OR 13/18 OR 21/30

A comma must be used to separate the items in any list, or the entire logical comparison must be specified.

The single exception to assuming OR as the list connector is operator NE. The implied connector in this case is AND because of the implied inversion of the equal condition for NE.

3.2.7 Predefining Conditional Expressions. Many GIPSY statements have an option for appending a conditional expression to the statement to control the data passed to it, or to control when the statement carries out the designated function. Some user applications have complex conditions in which it is desirable to simplify the user interface by predefining those complex conditionals and assigning a clearly understood name to them. To address requirements such as these an optional block structure called the LOGIC TABLE is provided. The LOGIC TABLE allows the creation of any number of conditional expressions as defined in section 3.2.6, and the assignment of a 1- to 12-character name. The syntax for this block structure is:

LOGIC TABLE.

<logic name 1> : <conditional expression 1>.

<logic name 2> : <conditional expression 2>.

.

.

<logic name n> : <conditional expression n>.

END.

The user supplied names for <logic name 1>, <logic name 2>, etc. should be meaningful names. The first character of the name must be alphabetic; it can contain no special characters and its length cannot exceed 12 characters. The colon is the assignment operator for attaching a name to the conditional expression on the right. Names defined in the logic table may be used in lieu of a conditional expression in any statement which allows a conditional expression, including the logic table itself. Once it has been defined, a logic name may also be treated as a logical comparison (see section 3.2.6) and thus may be joined to any other logical comparison via an AND or an OR connector even in the logic table definitions.

The logic table itself does not do anything. It is a passive tool for providing definitions which become active only as part of some other function. GIPSY does allow the user to sort the retrieved data. See section 4.2 for a discussion of the SORT capability.

The following example illustrates the definition of logic names in the LOGIC TABLE:

LOGIC TABLE.

CARRIERS: ANAME (1/2) EQ CV.

CRUISERS: ANAME (1/2) EQ CG.

SUBS: ANAME (1) EQ "S".

COMBAT-SHIPS: CARRIERS OR CRUISERS OR SUBS OR ANAME (1) EQ D, F, L.

COMBAT6FLEET: COMBAT-SHIPS AND FLEET EQ 6.

NONCOMBT6FLT: FLEET EQ 6 AND NOT COMBAT-SHIPS.

NONCOMBT7FLT: (FLEET EQ 7 OR TASK EQ "109") AND NOT COMBAT-SHIPS.

END.

We will use this LOGIC TABLE in future examples where conditional expressions are allowed. These particular logic names apply only to the data file conditional expressions. This example assumes that fieldnames ANAME, FLEET, and TASK have been previously defined in the current FDT.

Up to this point we have described the LOGIC TABLE to be used with the data file. There exists an option on the LOGIC TABLE statement for the index file. In this case the syntax is:

LOGIC TABLE FOR INDEX.

<logic name> : <conditional expression>.

·
·
·

END.

All the syntax and semantics described earlier apply here. However, only fieldnames belonging to the index file description may be used in the conditional expression. These logic names can be used only in statements or block structures operating on the index.

3.2.8 Data Retrieval. GIPSY has a powerful data retrieval capability for accessing the user's data file and pulling out the desired data. Graphic outputs and related statistical tabular reports are the targeted outputs. However, GIPSY is used frequently as a data query system in order to produce formatted reports or data base subsets. It is even effective against large hierarchical data structures with variable format record types.

The syntax for the data retrieval statements is:

RETRIEVE { FROM FILE } { IF <composite expression> }
 { FROM INDEX } { ALL }

where <composite expression> is

{
 <conditional expression>
 <type expression>
 <relative expression>
 <composite expression> { AND
 OR } <composite expression>
}

where <type expression> is

{
 <record_type>
 <record_type> CHANGES
 <record_type> COMPLETES
}

where <relative_expression> is

RECNR {
 GT
 GE
 EQ
 LT
 LE
 BT
}

<integer value>

The RETRIEVE statement is an optional statement. If one is not specified, all records in the file will be candidates for inclusion in the reports or graphic displays to be produced. The statement "RETRIEVE ALL." will also retrieve all records.

The data retrieval criteria may be composed of one of the two RETRIEVE statements: RETRIEVE FROM FILE or RETRIEVE FROM INDEX. If the FROM clause is not specified, FROM FILE is assumed. Use the FROM INDEX only if a GIPSY index file and the FDT for the index have been defined (for GFRC ISP files) or the indexes from the user's file have been correctly referenced in the file's FDT (for UFAS indexed or relative files and I-D-S/II indexed or integrated files). See sections 3.2.4.2 and 3.2.5.3 for the details and restrictions on building GIPSY indexes. For I-D-S/II integrated files if using FROM INDEX you must use the key specified in the index. This is obtained from the GET IDSII STRUCTURE statement. It will also be displayed at the end of the //IDSII statement and the //FDT statement. Typically, only users with very large files need be concerned about RETRIEVE FROM INDEX. The majority of applications simply use the data file RETRIEVE statement.

RETRIEVE FROM FILE IF or RETRIEVE IF causes GIPSY to test each record in the file against the specified <composite expression>. The <composite expression> may be composed of a <conditional expression>, a <type expression>, a <relative expression>, or two <composite expressions> joined by AND or OR. The syntax and semantics of a <conditional expression> are discussed in section 3.2.6.

A <type expression> is valid only for the I-D-S/II integrated file type and is made up of a <record_type> identifier, <record_type> identifier CHANGES, or <record_type> identifier COMPLETES. The <record_type> identifier is a unique four digit number assigned to each record type. It specifies the individual record types of the I-D-S/II integrated data file, which may be obtained by the GET IDSII STRUCTURE command and displayed by the //IDSII STRUCTURE interrupt command. When a <record_type> identifier is used alone, it acts as a logical field that is true when the current record is of that type. The <record_type> identifier CHANGES expression is true when the current record is of that type and the previous record is not of that type. The <record_type> identifier COMPLETES expression is true when the current record is of that type and the next record is not of that type.

A <relative expression> is valid only as retrieval criteria for the UFAS Relative file type. It consists of a RECNBR, a relational operator, and an integer value. RECNBR is a reserved word that is used only in a <relative expression> to identify a specific record. It may not be used as a data value. The valid relational operators are GT, GE, EQ, LT, LE, and BT. The operator BT requires a pair of integer values separated by a comma. The operator EQ allows comparison against a list of values separated by commas.

When the <composite expression> is satisfied (is true), that data record is retrieved for further processing; all data fields which are functionally referenced prior to the run are copied to the QDF to be used later in the

report or graphic display. A field is functionally referenced only if it appears on a statement which participates in the report or display building function. The command RETRIEVE ALL will cause every record be retrieved.

A simple retrieval statement:

```
RETRIEVE IF FLEET EQ 6.
```

might be used to retrieve all 6th Fleet data. Perhaps this statement retrieved too much data to derive the answer you are looking for, so let's make it more restrictive, say to select only combat ships in the 6th Fleet. That is,

```
RETRIEVE IF FLEET=6 AND (ANAME (1/2) EQ CV OR  
  ANAME (1/2) EQ CG OR ANAME (1) EQ S, D, F, L).
```

Note if the logic table in section 3.2.7 were available, this statement could have been coded RETRIEVE IF COMBAT6FLEET.

RETRIEVE FROM INDEX defines a test that is applied against the index field to determine which records in the data file are to be read. This function does not actually retrieve any data from the index. It uses the index in a manner similar to the index of a book to determine what part of the file will be considered. Effectively, GIPSY considers data records to exist only when the index conditional expression is satisfied. Therefore, RETRIEVE FROM INDEX does not see a data record until an index condition is satisfied. Without a RETRIEVE FROM INDEX, GIPSY must look at every record in the data base.

The bottom line is that a retrieval from a large indexed file which uses the GIPSY indexing capabilities is orders of magnitude faster than a plain retrieval.

The data qualification capability provided by the RETRIEVE statement may be augmented with the FIELD TABLE capability discussed in section 3.2.11. That capability allows data to be rearranged or supplemented after the data is read from file but before it is tested by the RETRIEVE mechanism. This data may be retrieved based on data modification or data not really in the data base.

Any field referenced on a RETRIEVE statement causes these fields to be included in the QDF and QDT. This may be inhibited by issuing the statement

```
SET AUTO INCLUDE OFF.
```

The data selection process is initiated by the command:

```
RUN.
```

Once the RUN command has been entered, GIPSY begins retrieving records from the users data file, field table moves are performed, and retrieval criteria is checked. If a record meets the retrieval criteria, the fields flagged for inclusion to the QDT are written out to the QDF. Following data selection, GIPSY places the user either in the statistical reports module (if BUILD

TABULAR REPORT was specified) or in the geographic module. Alternately, the user can control which module GIPSY ends up in by the command:

```
RUN TO { DISPLA  
        GEOMON  
        GDR }
```

You may interrupt the data retrieval process at any time by depressing the break key (or entering the appropriate device break function). GIPSY will stop reading the file and display its current file processing status (i.e., number, number retrieved, etc.) then allow you to terminate, continue, or to ignore the unread portion of the file and build the tabular report with only the data already retrieved. In the following example, the break key was depressed shortly after the "DATA Selection started" message. (Note the times.)

```
DATA Selection started at 1757 03.  
DATA Selection halted by BREAK KEY at 1757 09:  
RECORDS READ      RECORDS QUALIFIED    RECORDS IN ERROR  
    1343              53                  0
```

```
Enter "S" to process subset qualified above,  
    or "C" to continue qualifying records,  
    or "M" to continue with monitoring,  
    or "T" to terminate run.  C
```

```
DATA Selection ended at 1758 05:  
RECORDS READ      RECORDS QUALIFIED    RECORDS IN ERROR  
    2272              175                  0
```

Data selection monitoring can also be turned on automatically with the SET MONITOR command:

```
SET MONITOR { ON  
             OFF }
```

3.2.9 Arithmetic Expressions. Arithmetic expressions occur in many places in the GIPSY language. For this reason let us provide a basic description of an arithmetic expression here. When the term arithmetic expression is used in the remainder of this document it will have the syntax and semantics (meaning) described here. Arithmetic expression is any valid equation specifying an arithmetic or algebraic computation involving data fields from the file and arithmetic operators "*", "/", "+", "-", "(", and ")". The standard precedence of operations, including the use of parentheses to alter the precedence of calculations, is observed (i.e., parenthetical expressions are evaluated first, followed by expressions connected by * or /, then expressions connected by + or -).

3.2.10 Predefining Arithmetic Expressions. Arithmetic expressions may be predefined, given a name, and recalled later by referencing the name assigned

to it. These names are defined in a block structure called a math table. The math table is initiated by the statement MATH TABLE, followed by math names with associated arithmetic expressions and terminated with an END statement.

Specifically,

MATH TABLE.

<math name 1> = <arithmetic expression 1> .

<math name 2> = <arithmetic expression 2> .

.

<math name n> = <arithmetic expression n> .

END.

The <math names> must be unique within all user defined names (including field names) and are limited to 12 characters in length. These names may be used in subsequent statements to obtain the specified results.

For example:

MATH TABLE.

TOTAL-ROUNDS=RIFLE-CRATES*332+COLT-CRATES*429.

COLT-ROUNDS=COLT-CRATES*429.

RIFLE-ROUNDS=RIFLE-CRATES*332.

FIRE-POWER=(RIFLE-CRATES*332+COLT-CRATES*429)*RELIABILITY.

END.

The math table does for arithmetic expressions what the logic table does for conditional expressions. The math table itself is passive. Its presence does not affect processing. It is simply a means of avoiding redundant entry of an arithmetic expression.

3.2.11 Data Modifications. Most user data files were designed and implemented with no objectives relating to interactive graphic display. Most certainly not GIPSY graphic display. Consequently, occasions arise when data needed for effective graphic display or effective GIPSY usage does not exist in the data file or is not in a proper format for interactive graphic usage. GIPSY provides a mechanism to modify existing data fields and create new field and load data into them in an in-line extension of the data record. The new field may be created with the BUILD FDT block structure. Data is inserted into the new field or modified in an existing field via a mechanism called the field table. The field table is a block structure which allows data to be moved into a field, arithmetic computations to be done, data to be concatenated with other data, and conditional expressions to be used to control its processing.

The field table is primarily targeted toward the application programmer as opposed to the user-analyst. The user analyst who is unfamiliar with computer programming concepts can skip on to the next numbered section.

The syntax of the field table is:

FIELD TABLE [FOR <for clause>] [WHEN <composite expression>] .

where <for clause> is $\left\{ \begin{array}{l} \text{FILE} \\ \text{INITIAL} \\ \text{FINAL} \end{array} \right\}$

where <composite expression> is

$\left\{ \begin{array}{l} \text{<conditional expression>} \\ \text{<type expression>} \\ \text{<relative expression>} \\ \text{<composite expression>} \end{array} \right\} \left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\} \text{<composite expression>}$

where <type expression> is $\left\{ \begin{array}{l} \text{<record_type>} \\ \text{<record_type> CHANGES} \\ \text{<record_type> COMPLETES} \end{array} \right\}$

where <relative_expression> is RECNR $\left\{ \begin{array}{l} \text{GT} \\ \text{GE} \\ \text{EQ} \\ \text{LT} \\ \text{LE} \\ \text{BT} \end{array} \right\} \text{<integer value>}$

<fieldname> - <data to be assigned to fieldname> [IF <conditional expression>].

<fieldname2> - <data to be assigned to fieldname2> [IF <conditional expression>].

END.

The field table consists of the FIELD TABLE statement, one or more statements to assign values to a field, and an END statement to close out the block structure. Every record in the data file is processed against the field table prior to being made available for the RETRIEVE FROM FILE statement. (This means that the RETRIEVE statement can reference data based on the result of field table operations.) The WHEN <composite expression> clause controls overall use of the field table. A <composite expression> may be composed of a <conditional expression>, a <type expression>, a <relative expression>, or two <composite expressions> joined by AND or OR. The <conditional expression>

throughout the field table has the standard definition as provided in subparagraph 3.2.6.

A <type expression> is valid only for the I-D-S/II integrated file type and is made up of a <record_type> identifier, <record_type> identifier CHANGES, or <record_type> identifier COMPLETES. The <record_type> identifier is a unique four digit number assigned to each record type. It specifies the individual record types of the I-D-S/II integrated data file, which may be obtained by the GET IDSII STRUCTURE command and displayed by the //IDSII STRUCTURE interrupt command. When a <record_type> identifier is used alone, it acts as a logical field that is true when the current record is of that type. The <record_type> identifier CHANGES expression is true when the current record is of that type and the previous record is not of that type. The <record_type> identifier COMPLETES expression is true when the current record is of that type and the next record is not of that type.

A <relative_expression> is valid only as retrieval criteria for the UFAS Relative file type. It consists of a RECNR, a relational operator, and an integer value. RECNR is a reserved word that is used only in a <relative_expression> to identify a specific record. It may not be used as a data value. The valid relational operators are GT, GE, EQ, LT, LE, and BT. The operator BT requires a pair of integer values separated by a comma. The operator EQ allows comparison against a list of values separated by commas.

A data record will be processed against the body of the field table only if that record first satisfies the WHEN criteria. If the WHEN clause is not specified all records will be processed against the body of the field table.

The <for clause> is optional and may consist of the key word FOR followed by FILE, INITIAL, or FINAL. The FOR FILE clause is the default and is therefore not essential to the syntax. The FOR FILE clause indicates that the field table should be processed against all the records in the file. The FOR INITIAL clause causes the field table to be processed only for the first record during data retrieval. The FOR FINAL clause indicates that only the last record in the file should be included in the field table execution.

The body of the field table consists of one or more conditional assignment statements. For each assignment statement the <fieldname> on the left-side of the equal sign (=) designates the field to receive the value produced on the right by the data to be assigned to <fieldname>. This will be done if and only if the associated IF <conditional expression> is true.

Values assigned to fields via field table operations remain in the field until the contents of that field is modified by a new data record on another field table operation. The field table may be used to move data to the extended record area where it is only modified by field table operations. This allows data to be collected across any number of records for use as a single record.

The <data to be assigned to fieldname> may take any of the following forms:

- (1) <fieldname>
- (2) (<arithmetic expression>)
- (3) <literal value>
- (4) <math name>
- (5) { <fieldname> } : { <fieldname> } : { <fieldname> }
 { <literal value> } : { <literal value> } : { <literal value> }
- (6) TRUE
- (7) FALSE
- (8) \$ISPPTR
- (9) \$DATE
- (10) \$TIME

The first three cases above are self-explanatory. Note that arithmetic expressions must be enclosed in parenthesis, while literal expressions are enclosed in quotes.

In the fourth case $\langle \text{math name} \rangle$ is any previously defined Math Table entry.

In the fifth case the colon (:) serves as a concatenation operator.

Concatenation is the process of connecting two or more items. For example, assume that the field DAYS-ELAPSED contains the value 2.

Then the field table:

FIELD TABLE.

WHICH-DAY = "THE " ; DAYS-ELAPSED : "ND DAY".

WHICH-DAY = "THE " ; DAYS-ELAPSED : "RD DAY"; IF DAYS-ELAPSED EQ 3.

END.

would assign the string "THE 2ND DAY" to be field WHICH-DAY. The concatenation operator is also very useful in pulling together disjointed data from two or more fields into a single field. Let us assume that a data field, POINT, contains coordinates in the form 895959NXXXXX1793939E. These coordinates can be put in the standard GIPSY coordinate form with the field table entry:

GOOD-COORD=POINT (1/7) : POINT (13/20); IF POINT (8/12)="XXXXX".

The resulting value in GOOD-COORD would be 895959N1793939E whenever POINT contained XXXXX.

Cases 6 and 7 define values which may be assigned to logical fields. TRUE and FALSE are bit variables which may be used as a complete phrase in a conditional expression.

Case 8 is a special case. The pseudo field \$ISPPTR contains the ISP record pointer value needed to build an index file. This pseudo field can only appear in a field table on the right of the equal sign. This case is appropriate only when the users data file is an ISP file.

Cases 9 and 10 allow the user to store computer generated date and time into the specified field. The \$DATE option supplies a 6 character date of the format MMDDYY, whereas the \$TIME option provides a 4 character military time in the format HHMM.

The real utility of the field table is realized when the fields referenced are in the extended record area. The contents of the extended record area are altered only when the user specified field table assignments change them. Therefore, the extended record can be used to hold values until it is desirable to release that information with the data record.

The user may use any number of field tables. Every data record will be processed against each field table subject to the conditional expressions on the FIELD TABLE statement. Further discussions of the field table are presented in section 4.

3.2.11.1 User Subroutine Data Modifications. GIPSY data can also be modified by passing it to user built subroutines (see section 3.2.3.20). This is accomplished in the Field Table block structure by either field table assignments or stand alone calls.

The syntax for these statements is:

```
FIELD TABLE.  
  <fieldname> = $ <subroutine name> [( <arg1>, <arg2>, ... <arg n> )].  
  $ <stand alone subroutine name> [( <arg1>, <arg2>, ... <arg n> )].  
END.
```

The subroutine name is preceded by a dollar sign "\$". The subroutine arguments <arg1> thru <arg n> are field names, defined on the FDT, which either pass or receive data from the subroutine. The field table assignments \$ISPPTR, \$DATE, and \$TIME (see section 3.2.11) are in fact subroutines contained in the default subroutine library.

3.2.12. Defining and Executing a GIPSY Process What is a GIPSY process? A GIPSY process is any subset of GIPSY statements which can be grouped together to perform a particular function, generate a display, or simply display notes or comments to the user. The process is identified by an alphanumeric name which represents the group of statements. Two GIPSY statements are provided for defining and executing a GIPSY process. These are the DEFINE PROCESS and DO statements.

The DEFINE PROCESS statement identifies the beginning of the subset of GIPSY statements which make up the process and also provides for naming the process. The syntax for the DEFINE PROCESS is:

```
DEFINE PROCESS <process name>.
```

The DEFINE PROCESS statement is then followed by any number of GIPSY statements. The process is terminated by the command END PROCESS <process name>, or by entering a null response. For example, to define a process named "MAP" you could enter the following statement:

```
DEFINE PROCESS MAP  
MAP FILE WORLD2.  
SET GRID ON.  
DISPLAY MAP WINDOW CONTAINING USA.  
END PROCESS MAP.
```

The statements within the process named "MAP" are not immediately executed. They are essentially set aside as a canned procedure to be executed at some later time.

To execute the process named "MAP" requires a DO command to be issued. The syntax of the DO statement is:

```
DO <process name> .
```

When the DO command is issued, the named process on the DO statement will be invoked and the statements within the named process will be immediately executed. For example:

```
DO MAP.
```

will cause the MAP, SET and DISPLAY statements within the process named "MAP" to be executed and a display to be generated. After all the statements within the process are executed, control will be returned to the user for the next GIPSY command. If the DO command was issued from within a PCS, control will return to the next statement within the PCS.

The following examples are given to illustrate how one might use the define process:

```
DEFINE PROCESS REPORT.  
  FILE 837IDPX0/GIPSY/DATA/..TSTFIL  
  FDT 837IDPX0/GIPSY/DATA/..TSTFDT  
  BUILD TABULAR REPORT.  
  ROWS.  
    USE UNITTYPE.  
  COLS.  
    USE READINESS.  
  END.  
END PROCESS REPORT.
```

```
DEFINE PROCESS ARMY-RET.  
  RETRIEVE IF UNITTYPE EQ ARMY.  
  TITLE (SIZE LARGE) "ARMY READINESS".  
END PROCESS ARMY-RET.
```

```
DEFINE PROCESS NAVY-RET.  
  RETRIEVE IF UNITTYPE EQ NAVY.  
  TITLE (SIZE LARGE) "NAVY READINESS".  
END PROCESS NAVY-RET.
```

```
DEFINE PROCESS ALL.  
  RETRIEVE IF UNITTYPE EQ ARMY, NAVY.  
  TITLE (SIZE LARGE) "READINESS STATUS".  
END PROCESS ALL.
```

```

DEFINE PROCESS MENU.
  SET SIZE JUMBO.
  //NOTE      C9999 STANDARD REPORT CAPABILITY
  //NOTE
  //NOTE  IF YOU WANT THE ARMY REPORT ENTER "DO ARMY."
  //NOTE
  //NOTE  IF YOU WANT THE NAVY REPORT ENTER "DO NAVY."
  //NOTE
  //NOTE  IF YOU WANT BOTH REPORTS ENTER "DO BOTH."
END PROCESS MENU.

```

```

DEFINE PROCESS ARMY.
  DO REPORT.
  DO ARMY-RET.
  RUN.
  DISPLAY REPORT.
END PROCESS ARMY.

```

```

DEFINE PROCESS NAVY.
  DO REPORT.
  DO NAVY-RET.
  RUN.
  DISPLAY REPORT.
END PROCESS NAVY.

```

```

DEFINE PROCESS BOTH.
  DO REPORT.
  DO ALL.
  RUN.
  DISPLAY REPORT.
END PROCESS BOTH.

```

Above we have defined 8 processes named:

```

.  REPORT
.  ARMY-RET
.  NAVY-RET
.  ALL
.  MENU
.  ARMY
.  NAVY
.  BOTH

```

Each process has a predefined function. The statement may be entered either interactively or from a PCS file. Remember, none of the above statements have actually been executed up to this point. The following examples illustrate the use of the DEFINE PROCESS capability:

```
GIPSY PCS 837IDPX0/GIPSY/PCS/PCS-DEF
```

GIPSY Release 5.1 (1 November 89) **PRODUCTION SYSTEM**
For any problems or questions about this release contact the GIPSY SUPPORT
OFFICE AT (703) 695-3519, A/V 225-3519.

DEFINE PROCESS REPORT.

FILE 837IDPX0/GIPSY/DATA/..TSTFIL

FDT 837IDPX0/GIPSY/DATA/..TSTFDT

BUILD TABULAR REPORT.

ROWS.

USE UNITTYPE.

COLS.

USE READINESS.

END.

END PROCESS REPORT.

DEFINE PROCESS ARMY-RET.

RETRIEVE IF UNITTYPE EQ ARMY.

TITLE (SIZE LARGE) "ARMY READINESS".

END PROCESS ARMY-RET.

DEFINE PROCESS NAVY-RET.

RETRIEVE IF UNITTYPE EQ NAVY.

TITLE (SIZE LARGE) "NAVY READINESS".

END PROCESS NAVY-RET.

DEFINE PROCESS ALL.

RETRIEVE IF UNITTYPE EQ ARMY, NAVY.

TITLE (SIZE LARGE) "READINESS STATUS".

END PROCESS ALL.

DEFINE PROCESS MENU.

SET SIZE JUMBO.

//NOTE C9999 STANDARD REPORT CAPABILITY

//NOTE

//NOTE IF YOU WANT THE ARMY REPORT ENTER "DO ARMY."

//NOTE

//NOTE IF YOU WANT THE NAVY REPORT ENTER "DO NAVY."

//NOTE

//NOTE IF YOU WANT BOTH REPORTS ENTER "DO BOTH."

END PROCESS MENU.

DEFINE PROCESS ARMY.

DO REPORT.

DO ARMY-RET.

RUN.

DISPLAY REPORT.

END PROCESS ARMY.

DEFINE PROCESS NAVY.

DO REPORT.

```
DO NAVY-RET.
RUN.
DISPLAY REPORT.
END PROCESS NAVY.
```

```
DEFINE PROCESS BOTH.
DO REPORT.
DO ALL.
RUN.
DISPLAY REPORT.
END PROCESS BOTH.
```

```
DO MENU.
```

C9999 STANDARD REPORT CAPABILITY

```
IF YOU WANT THE ARMY REPORT ENTER "DO ARMY."
```

```
IF YOU WANT THE NAVY REPORT ENTER "DO NAVY."
```

```
IF YOU WANT BOTH REPORTS ENTER "DO BOTH."
```

```
>DO ARMY.
```

```
DATA Selection started at 1651 13
```

```
DATA Selection ended at 1651 14:
```

RECORDS READ	RECORDS QUALIFIED	RECORDS IN ERROR
23	5	0

1 OF 1

ARMY READINESS

	1	2	3
ARMY	1	1	3

In the preceding example, a PCS file was used to set up the processes to be executed and also invoke the process named "MENU". The statements which were used to set up the processes were echoed back to the terminal for illustrative purposes only. These statements can be made transparent to the user by having a //ECHO OFF statement as the first statement in the PCS. The following example illustrates this. NOTE: GIPSY would be invoked in the same manner as the first example (i.e., GIPSY PCS <cfid>):

C9999 STANDARD REPORT CAPABILITY

```
IF YOU WANT THE ARMY REPORT ENTER "DO ARMY."
```

```
IF YOU WANT THE NAVY REPORT ENTER "DO NAVY."
```


IF YOU WANT BOTH REPORTS ENTER "DO BOTH."

>DO BOTH.

DATA Selection started at 1655 16

DATA Selection ended at 1655 17:

RECORDS	RECORDS	RECORDS
READ	QUALIFIED	IN ERROR
23	6	0

1 OF 1

READINESS STATUS

	1	2	3	5
ARMY	1	1	3	0
NAVY	0	0	0	1

It should be noted that in the above examples, none of the statements within the processes invoked by a DO statements were echoed back to the terminal except for //NOTE statements within the process named "MENU". The echoing of information is a function of the //NOTE command; it is not a function of a process with the name of "MENU".

3.2.13. Executing GIPSY in the Batch Environment Occasions do arise when it is advantageous to execute GIPSY as a batch job. For the most part, executing GIPSY in the batch environment is used for large retrievals against a large data base to build other permanent files (QDF, GDS, etc.) or to generate standard, non-graphic reports (DISPLAY REPORT, PRINT) when the immediate time sharing response is not important but the extra resources available in batch are.

3.2.13.1. GIPSY Batch JCL. The JCL necessary for a GIPSY batch job initiated from time sharing is as follows:

```
CC
1      8      16

$      IDENT      <ident field>
$      PROGRAM    GIPSY, DUMP
$      LIMITS     05,16K,10K
$      PRMFL      **,R,R,LIBRARY/GIPSY/LOAD/BGIPSY
GIPSY
  <GIPSY statements>
```

DONE

If the statements are set up with line numbers, the numbers must be stripped when being RUN. The control cards needed to execute GIPSY in batch through a card deck are as follows:

```
CC
 1      8      16
$      IDENT      <ident field>
$      USERID      <userid $ password>
$      PROGRAM      GIPSY, DUMP
$      LIMITS      05,16K,10K
$      PRMFL      **,R,R,LIBRARY/GIPSY/LOAD/BGIPSY
GIPSY
  GIPSY statements
.
.
.
  DONE
$      ENDJOB
```

The GIPSY command statement can have the same options that are available under time sharing. These options are discussed in section 3.2.2.

3.2.13.2. GIPSY Batch Limitations. Most of what can be done with GIPSY in the time sharing environment can also be done in batch. The user who is interested in using GIPSY batch should be specifically aware of those statements and processes which are discussed below:

- a. The TSS Command - The TSS command allows the user to temporarily enter Honeywell's Time Sharing Subsystem from within GIPSY. This capability simply cannot be done in the batch environment and the job will abort if the TSS command is encountered during execution.
- b. The DONE Command - In the batch environment, the DONE statement must be the last GIPSY statement or the job will run until it exceeds execution time limits.
- c. The SAVE Command - The SAVE command is not functional in batch GIPSY. The RESAVE command will operate the same as it does in time sharing.
- d. Graphic Output Commands - The graphic output commands (DISPLAY MAP, DISPLAY BAR GRAPH, etc.) are executable in GIPSY batch. The SET GRAPHICS OFF option is implemented automatically. Graphic output can be sent to a file (see section 3.6, Picture Processing). The user can also process in batch up to the point where the graphic

output is displayed, and then resave the DAFC or GDS for future time sharing processing.

e. Error Correction - Interactive error correction is not possible in the batch environment. Results will be unpredictable if errors are encountered.

3.2.14. Parameterized Input Another of the user-friendly features in GIPSY is the ability to prompt the user for information whenever a pound sign is encountered in a GIPSY statement. When a "#" is encountered in a statement, such as "SET SIZE #.", GIPSY prompts the user with an equal sign "=" and the user is then expected to complete the statement (i.e., enter SMALL, J, LARGE, etc.). GIPSY then replaces the "#" with the input and continues processing.

If the statement containing the pound sign "#" is preceded with one or more //NOTE commands, the user can be informed as to what information is desired. If the pound sign is embedded in quotes, the "#" will be treated as a literal. Therefore, in the statement:

TITLE "#;#".

the pound sign (#) will be treated as a literal and the user will not be prompted, whereas:

TITLE #;#.

will cause the user to be prompted for two lines of title.

3.3 Statistical Reports

GIPSY produces a variety of graphic reports used to portray statistical and management data. It also produces an alphanumeric report which depicts this in the form of rows and columns. The basis for all of these reports is the Tabular Report which is described in the following discussion. Subsequent sections will describe how this report is built, how to display the reports, and finally how to enhance and modify them. The tabular report is the data structure from which all statistical graphs are generated. It is a formatted, matrix-oriented information structure with row headers down the left side, column headers across the top, and a matrix of data values in the body of the report. The numbers in the body of the report, the matrix, are the result of a cross tabulation of the occurrence of conditions defining the row and column headers. These numbers represent either counts or calculated values summed over each qualifying data record.

Any information summary occurring in the form of a matrix, or a cross tabulation of data items, or data conditions is a candidate for a GIPSY tabular report. Using GIPSY's tabular report capability is probably the most efficient and effective way to build a matrix where the data is counted or summed from a larger set of data records.

Before getting into the mechanics of building a tabular report, let us explain the relationship between the tabular report and the graphic report. Reflect back to your school days when you were given some equation of the form $y=f(x)$ and you were told to plot it. Let us suppose that the equation was $y=2x+2$. What you did was to set up a table assign an x-value and calculate its corresponding y. Associated pairs of x and y were produced until you were done. Then the graph was drawn (figure 3-4) from the (x,y) value listed below:

	X	Y
ROW 1	1	3
ROW 2	2	5
ROW 3	3	7
ROW 4	4	9
ROW 5	5	11
ROW 6	6	13
ROW 7	7	15
ROW 8	8	17
ROW 9	9	19
ROW 10	10	21

Now let us suppose that we had a family of lines to plot. Assume that they were $y^1=2x+1$, $y^2=3x+2$, $y^3=4x+3$, $y^4=x+4$. We could set up four (x,y) value lists, one for each equation.

Then we would proceed to assign a given x, and then calculate its corresponding y.

This process is repeated for each y, producing coordinates for four lines in four value lists.

x	y^1	x	y^2	x	y^3	x	y^4
1	3	1	5	1	7	1	5
2	5	2	8	2	11	2	6
3	7	3	11	3	15	3	7
4	9	4	14	4	19	4	8
5	11	5	17	5	23	5	9

But notice that there is a lot of redundant information. The same information is more efficiently displayed as:

x	y^1	y^2	y^3	y^4
1	3	5	7	5
2	5	8	11	6
3	7	11	15	7
4	9	14	19	8
5	11	17	23	9

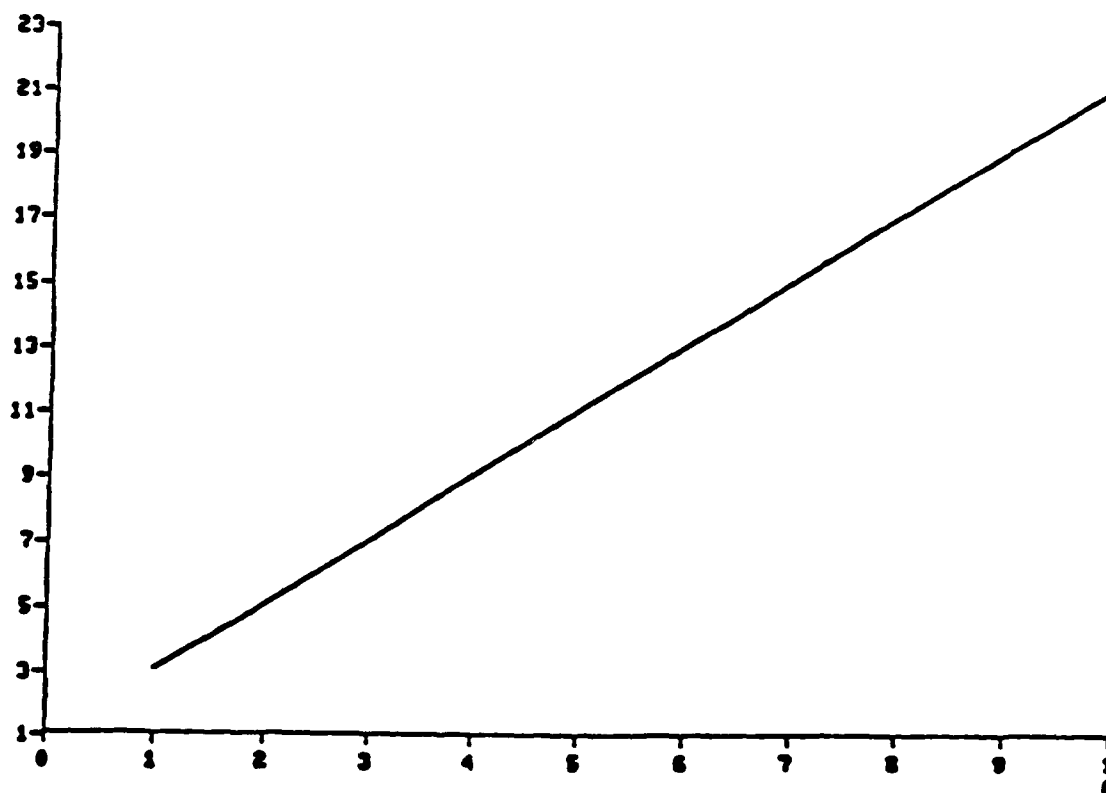


Figure 3-4. Graph of $y=2x+1$

This form of the (x,y) vector list is readily recognizable as a tabular report. This form makes it very convenient to display graphs of y^1 , y^2 , y^3 , and y^4 .

This is the concept GIPSY uses. In fact, this form together with classifications and titles, comprises a report on the Graphic Data Set (GDS) for management/statistical graphs.

The equation illustrated in the foregoing discussion limited the x values to numbers which are assigned to variables in an equation. GIPSY does not have this restriction since the tabular report is built from the user's data rather than calculations using an equation.

Bargraphs, histograms, line graphs, point graphs, and step graphs, use either the row or column headers as labels on the x-axis, and each separate item is equally spaced along the x-axis in order of appearance without regard to the numeric value (even if the item is numeric) of the x-axis label. If the row headers are taken as the x values, the y values are taken from the selected column of values; if the column headers are taken as the x values, the y values are taken from the selected row of values.

3.3.1 Building Statistical Reports. The building of a statistical or tabular report is a block-structured process. The block structure begins with the command BUILD TABULAR REPORT and is terminated with END. The general syntax form is:

```
BUILD TABULAR REPORT.  
    <row and column definitions>  
    [<category and section definitions>]  
END.
```

A Tabular Report may be created with:

- o rows and columns
- o rows, columns, and categories
- o rows, columns, and sections
- o rows, columns, categories, and sections.

3.3.1.1 Row and Column Definitions. The tabular report must have rows and columns. The row and column definitions are specified using a combination of one or more of the following four matrix building processes:

- o USE - uses the contents of a data field in the file to generate row and/or column headers.
- o SELECT - selects the contents of a number of fields from the file defining each field as a row and/or column in the report.

- o RANGE - ranges values and fills in the ranges from the data file.
- o EXPLICIT - specifically defines the name, calculation (or countings) and conditions under which data is to be entered in each element of the row and/or column.

The row and column specification can use the same or dissimilar processes. These are mapped together to build the tabular report. Upon recognizing the command BUILD TABULAR REPORT, GIPSY puts itself in a mode to accept ROW definitions, COLUMN definitions, or an END statement. The END statement will take GIPSY out of the tabular report building process. Both row and column specifications must be completed in order to generate a tabular report. The vectors are defined by specifying:

COLUMNS[:]	}	USE [UNSORTED]<list of fields>[<conditional/arithmetic expression>]
		SELECT <select list of fields>[<conditional/arithmetic expression>]
ROWS[:]		RANGE <range definitions> [<conditional/arithmetic expressions>]
		EXPLICIT <explicit definition>
		* <explicit definition>
		CALCULATED DATA = <arithmetic expression> [IF <conditional expression>]

Again, each tabular report must have a ROWS and COLUMNS definition as a minimum. A colon (:), a period (.) or a space may be used to separate the vector mode (ROWS,COLUMNS) from its vector definition.

3.3.1.1.1 USE Process. The USE statement tells GIPSY to use the contents of each of the specified fields to create the vector (row or column) headers. If a value extracted from the specified field changes and has not already been identified as a vector header, GIPSY will flag this item as a possible candidate for the next vector header. If the value has occurred previously from the same field, and is already a vector header, the vector containing the header value is flagged as a candidate vector to have the corresponding data vector updated. Note that we said that it is a "candidate" for update. Neither the headers nor the body of the report is updated until GIPSY has candidate vectors for both rows and columns.

In the USE statement, GIPSY's default is to automatically sort the vector headers in ascending order. If UNSORTED is specified, the headers will be added to the tabular report in the order encountered within the users data file.

The <list of fields> on the USE statement is expanded as a list of field names with optional prefixes separated by commas in the form:

```
<fieldname> ["<prefix literal>"] [,<fieldname2> ["<prefix literal>"]...]
```

If a <prefix literal> is specified, it is used as a prefix for the vector header value extracted from that field contents.

Note that all the USE command does is to build and/or identify the vector headers. The actual contents of the data portion of the tabular report are obtained from another source such as CALCULATED DATA or an explicit vector definition for another vector mode (for a detailed explanation of CALCULATED DATA see section 3.3.1.1.5). If USE is specified for both row and column, the result is a cross tabulation of the contents of the two specified fields. The body of the report will be composed of a count of the joint occurrences of row and column items.

The optional clause defining the <conditional/arithmetic expression> allows an arithmetic expression and/or a conditional expression to be attached to each statement within the tabular report block structure. The general form of this clause to be appended to the statement is:

- o - [SUM] <arithmetic expression> [IF <conditional expression>]
- o IF <conditional expression>

If an arithmetic expression is included, the value of the arithmetic expression is added to each entry created by the associated <list of fields>. The IF clause determines when data records will be made available to the USE statement.

For example, let's assume that you have a data file as follows:

STATE	NAME	CODE	LENGTH	CAPACITY	SURFACE	PERSONNEL
AL	MUSCLE SHOALS	C	6693	C130	CONCRETE	150
AL	BESSEMER	C	3800	A1	CONCRETE	75
AL	FOLSOM FIELD	C	5120	OV1	CONCRETE	110
AL	SHELBY COUNTY	C	3800	T38	ASPHALT	58
FL	ST.AUGUSTINE	C	6947	C123	ASPHALT	120
FL	JACKSONVILLE	J	8000	C141	CONCRETE	352
FL	SANFORD	C	8000	P3	ASPHALT	161
GA	ATHENS	C	4989	C131	ASPHALT	81
GA	BUSH FIELD	J	8001	C135	ASPHALT	61
GA	LEE GILMER MEM	C	3998	A1	ASPHALT	92

Suppose you need a report showing the names of airfields who have PERSONNEL greater than 90, the name of the state it resides in, and its corresponding surface. The following commands would produce such a report:

```
BUILD TABULAR REPORT.
  ROWS.  USE STATE IF PERSONNEL GT 90.
  COLUMNS.  USE NAME, SURFACE.
END.
```


The resulting report would be:

	NAME	SURFACE
AL	MUSCLE SHOALS	CONCRETE
AL	FOLSOM FIELD	CONCRETE
FL	ST. AUGUSTINE	ASPHALT
FL	JACKSONVILLE	CONCRETE
FL	SANFORD	ASPHALT
GA	LEE GILMER MEM.	ASPHALT

There is no need to presort the data since the matrix is dynamically built and sorted on the row and column headers in ascending order when the report is completed. Sorting of the report headers automatically occurs on any vector mode (row or column) which was constructed with the USE process, unless the UNSORTED option is used.

3.3.1.1.2 SELECT Process. A second report building function is accomplished by the SELECT statement. When the file contains several numeric fields which can be lifted as rows or columns in the tabular report, the SELECT statement may be used to obtain those values from the file and include the fieldname in the tabular report as vector headers.

SELECT is typically paired with a USE statement to build a tabular report. A simple example based on our airfield file would produce the following report.

```
BUILD TABULAR REPORT.  
  ROWS.  USE STATE.  
  COLUMNS.  SELECT PERSONNEL.  
END.
```

```
      PERSONNEL  
  
AL      393  
FL      633  
GA      234
```

Referring to the data file, the SELECT statement causes all PERSONNEL to be summed for each corresponding STATE.

SELECT has an implied arithmetic process. Consequently, if SELECT is specified on both rows and columns, the summation process will be done for each vector mode. If a sum phrase (arithmetic expression) is included, its value is computed and also added to the vector. An IF clause determines when data is to be made available for the SELECT process.

```
BUILD TABULAR REPORT.  
  ROWS.  USE CODE.  
  COLUMNS.  SELECT PERSONNEL IF LENGTH GT 5500.  
END.
```

PERSONNEL

C	431
J	413

In the above example and again referring to our data file, all personnel are summed for the vector CODE whose runway lengths are greater than 5500.

If the fieldname is to be used as a column or row header, an alternate name can be supplied. The syntax for the SELECT <list of fields> is:

```
<fieldname1> ["<alternate header1>"]  
[,<fieldname2> ["<alternate header2>"] . . .]
```

This statement causes GIPSY to use the fieldname, or alternate header, if specified, as the header of a row or column. The alternate header must appear on the same line as the fieldname.

3.3.1.1.3 RANGE Process. The third report building function is accomplished by the RANGE statement. This statement allows you to establish sets of numeric and alphanumeric ranges into which data will be processed. The ranges are established as vector (row or column) headers and data is processed into the data portion of the report in a manner similar to the USE, except that all headers are predefined by the RANGE statement.

The general forms of the range definitions are:

- o RANGE <fieldname> FROM <lower numeric value>
TO <high numeric value> [BY <numeric increment>].
- o RANGE <fieldname> FROM <alphanumeric value> TO <alphanumeric value>.

The FROM-TO-BY portion of the statement above may be repeated as often as desired as long as a comma is used to separate each set. Both forms may be freely mixed within one range statement. The statement must end with a period.

The FROM, TO, and BY arguments are used to define sets of ranges against which the contents of the field identified by fieldname will be tested to determine which range will be the candidate vector.

Using our data file, we may want to see a personnel breakdown by state. The following commands would produce an appropriate report:

```
BUILD TABULAR REPORT.  
ROWS. USE STATE.  
COLUMNS. RANGE PERSONNEL FROM 0 TO 60, FROM 61-120,  
FROM 121-180, FROM 181 TO 400.  
END.
```

	0-60	61-120	121-180	181-400
AL	1	2	1	0
FL	0	1	1	1
GA	0	3	0	0

Note that RANGE has no implied computations of data for the numeric portion of the matrix, hence the data portion may contain counts, the results of an arithmetic expression or some value calculated by another vector mode (e.g., SELECT).

3.3.1.1.4 EXPLICIT Process. If none of the three foregoing processing methods are appropriate, then an explicit definition of each row and/or column can be used. The explicit definition allows you to specify the name of the vector, the computations for that vector and the conditions under which that vector will be selected for processing or as a candidate vector. An asterisk following the declarative ROWS indicates an explicit definition will follow. The defined vectors are retained in the tabular report in the order specified.

The general form of the statement for explicit definition is:

<vector header> [- <arithmetic expression>] [IF <conditional expression>].

The <vector header> is 1 to 36 characters and is used as headers for the specified row or column. If the header contains special characters, it must be enclosed in quotes. Quotes are not required at other times but their use is strongly recommended to avoid possible conflict with GIPSY command words. The arithmetic expression is as we defined earlier. Recall that it may be a name assigned to an entry in a previously defined math table.

The <conditional expression> can either be used by itself or in conjunction with the <arithmetic expression>. It follows all the rules and conventions defined for the RETRIEVE statement in section 3.2.8, including the use of logic names. If a <conditional expression> is not specified, every data record becomes a candidate for this vector definition.

Each time a data record causes both the row and column to have a candidate vector, the value of the arithmetic function is computed and summed into the proper entry in the matrix. If an arithmetic expression is not specified, the resulting value will be the counts of the occurrence of the specified condition as a function of the other vector mode definition(s).

If no <conditional expression> is specified, the item always becomes a candidate vector.

As many vector headers as desired may be specified subject to the limitations imposed by your site on the size of TSS modules.

A tabular report using an explicit definition on columns and a USE on rows would appear:

BUILD TABULAR REPORT.

ROWS. USE UNIT-TYPE.

COLUMNS.*

"NR ARMY UNITS"IF SERVICE EQ ARMY AND STATUS EQ READY.

"NR ARMY TROOPS"-SUM ACTLSTRNTH; IF SERVICE=ARMY AND STATUS EQ READY.

"NR NAVY UNITS"IF SERVICE EQ NAVY AND STATUS EQ READY.

"NR NAVY TROOPS"-SUM ACTLSTRNTH;IF SERVICE EQ NAVY AND STATUS EQ READY.

UNDERSTAFFED=DIFRNC IF SERVICE EQ ARMY, NAVY AND STATUS EQ READY.

END.

In the foregoing example, several permutations of the explicit definition are used, including the use of a math name. It is assumed that a Math Table was previously entered in which the equation "DIFRNC" was defined.

Each vector mode (ROW or COLUMN) may be specified using a single process (e.g., USE) or as many combinations of processes as desired to define the report. The integrity of the data from each definition is maintained and the data is collected together. For example, if a SELECT and USE were used to build the rows of a tabular report all the SELECTS will be collected in one set of vectors followed by all the USE vectors.

Conditional expressions used here have the same syntax and semantics as described in section 3.2.6. If a conditional expression is associated with any statement or clause that conditional expression is evaluated prior to attempting to perform the associated process.

3.3.1.1.5 Specifying Calculated Data Values. Another method for providing values in the tabular report is to specify what is called a CALCULATED DATA statement. CALCULATED DATA is a mechanism for specifying an equation whose result is to be added to the proper element in the tabular report for the row or column being processed. The data element updated is a function of the definition for the vectors. However, the CALCULATED DATA function is a global function. A CALCULATED DATA statement is not required. It is simply one of the methods of specifying data computations to be applied to the tabular report. If CALCULATED DATA is specified, its value is always added to every qualifying entry in the tabular report for every qualifying record regardless of whether additional computations will also be applied to the value. The syntax for this statement is:

CALCULATED DATA - <arithmetic expression> [IF <conditional expression>].

The CALCULATED DATA value is calculated anew for each data record retrieved. As an example, let's assume that we are working with an ammunition data file which contains the number of crates of Colt 45 rounds (COLT-CRATES) and the number of rifle rounds (RIFLE-CRATES) being sent to each company. Assume further that we need to build a tabular report whose data elements are

measured in rounds. Knowing that there are 322 rounds in each rifle crate and 429 in each Colt crate, the problem is easy to solve. We simply tell GIPSY:

CALCULATED DATA=RIFLE-CRATES * 322 + COLT CRATES * 429.

Since CALCULATED DATA is a record function, the proper value will be calculated for each record to be used in building the tabular report. The tabular report definition itself identifies the specific vector to which these values will apply.

If a <conditional expression> is included on the CALCULATED DATA statement, the value of the arithmetic expression is computed only when the defined condition is satisfied; otherwise, the value is undefined and a zero value is assumed. If no conditional expression is used the value is always calculated.

3.3.1.2 Category and Section Definitions. Category and section definitions are optional. They are used to expand the tabular report into a four dimension report by adding a structure above the columns called categories and above the rows called sections. Observe the illustration below.

	CATEGORY I		CATEGORY II	
	COL A	COL B	COL A	COL B
SECTION I				
Row 1	1	4	2	7
Row 2	2	5	3	9
Row 3	3	6	5	11
SECTION II				
Row 1	2	8	10	40
Row 2	4	10	20	50
Row 3	6	12	30	60

Note the symmetry of the row headers within the sections and the symmetry of the column headers under the categories. Categories and sections are defined using the same set of capabilities used to define rows and columns. Every process defined for row and column in section 3.3.1.1 applies individually for each section and category respectively. Any of the four report building processes can be used on categories and sections regardless of the process used to build the rows and columns.

The syntax and semantics of USE, SELECT, RANGE and * (EXPLICIT) are identical to that of rows and columns except that they apply to the categories and sections.

3.3.1.3 Defining Multiple Reports. During data retrieval, GIPSY uses the BUILD TABULAR REPORT definitions to create a matrix of rows and columns which is stored on a file called the Graphic Data Set (GDS). All previous discussion has dealt with a GDS containing only a single matrix (tabular report). GIPSY also has the capability of producing a GDS which contains multiple reports. This feature is controlled by the BREAK command.

The syntax for the BREAK command is:

```
BREAK { WHEN <conditional statement> }
      ON  <field name> }
```

```
[ [;ID=[(SIZE <size option>) ] { <fieldname> } [ { <fieldname> } ] ... ]
  "<literal>" ]
```

The BREAK statement affects the processing of QDF records in building the matrix. Using the WHEN option will cause the current report building process to be terminated and a new report building process to be started whenever the conditional expression is met. When using the ON option a new report will be generated each time the <field name> changes.

The BREAK statement must be entered prior to the RUN command. The ID option is used to provide a unique label to each report. When the report is displayed, the ID will appear directly beneath the title.

Consider the following data which contains information on equipment possessed by various services.

<u>SERVICE</u>	<u>EQUIPMENT</u>	<u>HOME</u>	<u>DEPLOYED</u>
NAVY	SHIPS	12	90
NAVY	PLANES	35	70
ARMY	TANKS	39	80
NAVY	SUBS	13	60
ARMY	HELICOPTERS	29	30
NAVY	SHIPS	14	20
ARMY	HOWITZERS	42	70

Figure 3-5 shows how the BREAK command is used to create a separate report for each service. Note that the row headers are different in each report. The ACCESS command is used to load each report (see section 3.3.6).

3.3.2 Displaying Statistical Reports. You are now ready to actually produce and display the tabular report from that portion of your data file which meets the criteria defined in the RETRIEVE statement. (If no RETRIEVE statement is specified the default is to retrieve all data records.) To initiate this action the statement

RUN.

is issued. At this point, GIPSY will read the data file last specified in the FILE statement, retrieve the records specified by the retrieval criteria, and output the resulting data on a temporary file called the Qualified Data File (QDF). When the retrieval portion is complete, GIPSY will display a message showing the number of records read and retrieved.

GIPSY Release 5.1 (01 November 89) **PRODUCTION SYSTEM**
For any problems or questions about this release contact the GIPSY SUPPORT
OFFICE AT (703) 695-3519, A/V 225-3519.

FILE BRKFILE
FDT BRKFDT
BUILD TABULAR REPORT
ROWS: USE EQUIPMENT.
COLS: SELECT HOME, DEPLOYED.
END.
BREAK WHEN SERVICE CHANGES;ID U.S. ", SERVICE.
SORT ON SERVICE.
RUN
Data Selection started at 0553 46
Data Selection ended at 0553 49:
RECORDS RECORDS RECORDS
READ QUALIFIED IN ERROR
7 7 0

>DISPLAY REPORT.

U.S. ARMY

	HOME	DEPLOYED
HELICOPTERS	29	30
HOWITZERS	42	70
TANKS	39	80

>ACCESS REPORT 2.
>DISPLAY REPORT.

U.S. NAVY

	HOME	DEPLOYED
PLANES	35	70
SHIPS	26	110
SUBS	13	60

Figure 3-5. Example of BREAK Command

Each record placed on the QDF is processed against the tabular report block structure. When all records have been processed, the tabular report is output on a temporary file called the Graphic Data Set (GDS). Then GIPSY places you in the graph and report output mode; the prompt character is displayed and the keyboard is opened for commands. If the commands are coming from a PCS file instead of the keyboard, GIPSY will resume processing from the PCS file rather than opening up the keyboard for commands.

The different types of reports that GIPSY will display for you are:

- o Tabular Report
- o Bar Graph
- o Histogram
- o Point Graph
- o Line Graph
- o Curve Graph
- o Step Graph
- o Gantt Chart
- o Pie Chart

3.3.2.1 Tabular Report. The command DISPLAY REPORT is issued whenever the tabular report is to be displayed. The tabular report (matrix) is automatically formatted to ensure a neat appearance of the data. If a TITLE command was issued, the titles are centered, trailing zeros are dropped, decimal points are inserted as appropriate, and all the columns are made symmetric with respect to each other. The report is immediately displayed one page at a time. If the report is a multipage report, all the rows will be displayed followed by all the columns. Assuming a 9-page report, the pages would be displayed and numbered in the sequence shown in figure 3-6. If the report contains sections and categories, the process illustrated in figure 3-6 is repeated for each section, and then the section process is repeated for each category. If a section or category is less than a page GIPSY will pack as many complete sections and/or categories per page as will fit. It will not place a partial section or category on the same page with a different one.

The syntax of the DISPLAY REPORT command is:

```
DISPLAY REPORT [ ( [ STANDARD ] [ MAXDEC <n> ] [<vector list>] ) ]
                  [ COMPRESSED ] [ FIXDEC <n> ]
```

DISPLAY REPORT may be abbreviated to "D R" or "DR". The report is normally displayed using the default STANDARD form in which spacing for all columns is symmetric. COMPRESSED overrides the column symmetry function by removing unnecessary spaces between columns to get more data on each page. An entire page will be formatted and displayed. When the page is full, the keyboard will be opened to accept new commands (a carriage return will continue current processing). An end of page condition is indicated by sounding the bell rather than displaying the prompt character.

COLUMNS →

ROWS ↓

1 of 9	4 of 9	7 of 9
2 of 9	5 of 9	8 of 9
3 of 9	6 of 9	9 of 9

Figure 3-6. GIPSY Page Numbering

GIPSY calculates the number of integer digits and decimal places for each column of the tabular report individually. All values are retained internally as floating point numbers. If no value in the column contains a fractional value, the column of data is displayed as a set of integers. If the column contains any fractional value, the number of integer digits and decimal places are determined by the largest value and the number of digits required to represent the longest fractional part in the column. This length may be reduced if the space is needed to display the integer portion of the number. GIPSY's field width for a column of numbers will accommodate eight digits. If any value in the column cannot be accommodated in this space, GIPSY will revert to the scientific notation for displaying that column. In any case the entire column will always be displayed according to the same format. If the user wishes to have full control of the format of textual reports, then see section 7 for a complete explanation of GIPSY's Generalized Data Reports. In most cases the GIPSY computed format for the column of numbers is adequate. However, the optional parameter MAXDEC and FIXDEC allows the GIPSY format to be overridden. MAXDEC <n> declares that no column in the tabular report shall have more than <n> digits to the right of the decimal point. GIPSY will continue to decide for those columns which will have less than the specified value of <n>. The parameter FIXDEC declares that every entry in the tabular report shall be displayed with the fixed number of decimal places as specified. The command:

DISPLAY REPORT (FIXDEC 0).

will cause the entire report to be displayed as integers regardless of the fractional parts. Every number will be rounded to the nearest whole number. Reports whose values represent percentages are frequently displayed with FIXDEC 2 as the optional parameter. This causes all values to be displayed in the form <xx.ff> where <ff> represents two fractional digits and <xx> represents up to six digits, as required, to display the integer portion of the number.

A command such as:

DISPLAY REPORT (MAXDEC 3).

will cause a maximum of 3 decimal places to be printed, but will not add decimal places to a set of values having less than 3 decimals (i.e., a column of integers).

In its normal default mode of operation, GIPSY will display the entire defined report. For a given display sequence, however, it may be desirable to display only a limited subset of the entire tabular report. The last option on the DISPLAY REPORT command allows a <vector list> which identifies specifically which row, column, section, and/or category is to be included in the report displayed. The detailed syntax of the <vector list> as used in the DISPLAY REPORT syntax definition is of the form:

COLUMNS <name list>
ROWS <name list>
CATEGORIES <name list>
SECTIONS <name list>

where <name list> is a list of vector names from the identified vector mode. This list of vector names may be a series of vector names separated by commas, lists of vectors identified by a "thru" list, or combinations thereof. The "thru" list is in the form:

<first vector desired> THRU <last vector desired>

If multiple vector modes are specified, each mode's <vector list> must be terminated by a semicolon. If the <last vector desired> occurs ahead the <first vector desired> in the tabular report, the order of appearance of the vectors in the "thru" list will be reversed in the display.

The report displayed will consist of only the specified vectors; the vectors will be displayed in the order specified in the vector list. (If no vectors are identified all vectors will be included.) This display modification will apply only for the duration of that DISPLAY REPORT command. This feature allows the report to be dynamically reconstructed for display purposes without disturbing the actual tabular report. For example, given a report with rows named R1, R2, ..., R50 and columns named A, B, C, ..., Z (appearing in that order) the command:

DISPLAY REPORT (ROWS R50 THRU R1; COLS Z,B,C,G THRU K,M,R THRU Y,A).

will cause the report to be displayed with the rows on reverse order (first row will now be R50); columns D,E,F,L,N,O, and P will not be displayed and columns Z and A will be interchanged. Recall that the vectors will be displayed in the order specified. If it is desirable to cause these same limitations to be imposed on subsequent display actions, these limitations should be defined via the LIMIT statement described in section 3.3.3.1.1.

The following examples of tabular reports use the specifications for reports as described above:

Example 1: USE/USE

The user's file contains a field called REPORTTYPE whose contents are report types, such as DISUM, SPIREP, SITREP, and OPREP3; and a non-numeric field called CLASSIF whose contents are security classification symbols, such as T (Top Secret), S (Secret), and C (Confidential). The user file containing these data values is represented as follows:

<u>REPORT- TITLE</u>	<u>REPORT- TYPE</u>	<u>CLASSI- FICATION</u>	<u>NR- PAGES</u>
EXERCISE ALPHGIPSY	DISUM	S	30
EXERCISE LOW HEELS	DISUM	T	20
REPORT ON EXERCISE UTILITY	SPIREP	S	10
MESSAGE ANALYSIS REPORT 385	OPREP3	S	10
REFLECTIONS ON LOW HEELS	SITREP	S	10
MUSICAL CHAIRS 995	OPREP3	C	10
SITUATION STATUS	SITREP	S	10
CRISIS ACTION PLAN	OPREP3	S	10
EXERCISE ANALYSIS PLAN	SITREP	T	10
EVALUATION 385	SPIREP	S	10
GRAPHIC SUMMARY OF LOW HEELS	SPIREP	S	10
EXERCISE ANALYSIS GUIDELINES	DISUM	C	10
RESULTS OF ALPHGIPSY	SPIREP	C	10
RECOVER OF LOSSES	SPIREP	C	10

The user wishes to construct a report showing the distribution of classified reports by report types. Since the fields REPORTTYPE and CLASSIF contain information suitable for row and column headers and we want a count of occurrence of each report by classification, we will specify a USE on both rows and columns. The request we enter is:

```
FILE 837IDPXX/USER/DATA/FILE
FDT 837IDPXX/MYFDT
BUILD TABULAR REPORT.
COLUMNS. USE CLASSIF.
ROWS. USE REPORTTYPE.
END.
RUN.
```

The tabular report resulting from the USE/USE option appears as follows:

	C	S	T
DISUM	1	1	1
OPREP3	1	2	0
SITREP	0	2	1
SPIREP	2	3	0

Example 2: RANGE with USE

Two forms of the RANGE option are illustrated below using an input file containing fields called NAME, JOBCODE, and SALARY. Each field contains data values related to the distribution of incomes among personnel in different jobs. The file contains the following data:

<u>NAME</u>	<u>JOB</u> <u>CODE</u>	<u>SALARY</u>	<u>TYPE-WORK</u>
JOHNSON, LARRY	123	10000	BLUE-COLLAR
SMITH, RICHARD	124	12000	BLUE-COLLAR
POWELL, WILLIAM	125	14000	BLUE-COLLAR
STEVENS, CAROLE	123	11500	BLUE-COLLAR
NELSON, ROBERT	124	14000	BLUE-COLLAR
VAUGHAN, NANCY	125	15000	BLUE-COLLAR
ALEEN, ANTHONY	125	15000	BLUE-COLLAR
MCPHERSON, ALISTER	123	12000	BLUE-COLLAR
WILLIAMS, ROGER	124	13000	BLUE-COLLAR
TUDOR, ELIZABETH	123	10500	BLUE-COLLAR
WALKER, JAMES	BOSS	55350	SUPERVISOR

The following language statements create a matrix containing the count of blue-collar employees by JOBCODE which qualify for the indicated salary range intervals:

```

FILE COMPANY/PAYROLL
FDT PAYROLL/DESCRIPT
RETRIEVE IF TYPE-WORK="BLUE-COLLAR".
BUILD TABULAR REPORT.
COLUMNS. RANGE SALARY FROM 10000 TO 11999, FROM 12000 TO
      13999, FROM 14000 TO 15999.
ROWS. USE JOBCODE.
END.
CLASS UZZ.
TITLE "BLUE COLLAR JOB CODE WAGE DISTRIBUTION".
RUN

```

A tabular report for the resulting matrix would appear as follows:

```

UNCLASSIFIED

BLUE COLLAR JOB CODE WAGE DISTRIBUTION

      10000-11999      12000-13999      14000-15999
123          3          1          0
124          0          2          1
125          0          0          3

```

UNCLASSIFIED

Note that since the ROWS were specified by USE, the labels were automatically sorted in ascending sequence.

The above RANGE statement could be expressed more simply with the BY clause added to the statement format, as follows:

COLUMNS. RANGE SALARY FROM 10000 TO 15999 BY 2000.

By adding the statement:

CALCULATED DATA - SALARY.

to the request, the tabular report produced would contain the total amount of salary paid for each job code within the designated salary ranges.

Example 3. USE/SELECT

The SELECT with USE option creates a matrix whose columns are labeled with the unique values from the specified field, and the rows are labeled with the specified fieldnames. GIPSY sums the contents of each selected field and places the total in the appropriate matrix location. Each field is handled separately.

An example of the USE/SELECT option statements is shown below:

```
FILE FLITEFIL
FDT FWTDT.
BUILD TABULAR REPORT.
  COLUMNS. USE FLIGHTNUMBER.
  ROWS. SELECT NUMPASSENGER.
END.
RETRIEVE IF DAY LT 30 AND MONTH EQ 4,5.
RUN
```

The statements specify a matrix with column labels taken from the contents of the FLIGHTNUMBER field of the records in a user file. These values are FLT604, FLT725, and FLT821. The row label is NUMPASSENGER, which is the name of the specified field. The matrix resulting from these statements contains the sum of the number of passengers, respectively, for each of the flights of FLT604, FLT725, and FLT821.

The user file containing the data values is represented as follows:

<u>FLIGHTNUMBER</u>	<u>NUMPASSENGER</u>	<u>MONTH</u>	<u>DAY</u>
FLT604	120	4	29
FLT725	100	4	29
FLT821	150	4	29
FLT604	200	4	30
FLT725	160	4	30
FLT821	180	4	30
FLT604	100	5	1
FLT725	150	5	1
FLT821	100	5	1

The user could display the resulting matrix and obtain a tabular report as follows:

	FLT604	FLT725	FLT821
NUMPASSENGER	220	250	250

It is possible to specify an alternate name to replace the fieldname as a vector label in the SELECT option:

```
BUILD TABULAR REPORT.
COLUMNS. USE FLIGHTNUMBER.
ROWS. SELECT NUMPASSENGER "APRIL VOLUME".
END.
RETRIEVE IF MONTH EQ 4.
```

The matrix resulting from these statements would contain the sum of the number of passengers for April. A tabular report for the matrix would appear with the optional row label:

	FLT604	FLT725	FLT821
APRIL VOLUME	320	260	330

Example 4. USE/USE with Arithmetic Expression

The following example illustrates adding an arithmetic expression to a USE statement:

```
RETRIEVE IF CINC EQ "3 N", 3N AND TOF BT 060001/082359
AND MONTH EQ JUNE.
BUILD TABULAR REPORT.
ROWS. USE REPORTTYPE - TEXTLINES.
COLUMNS. USE CLASSIFICATION.
END.
```

The user file containing the data values is represented as follows:

<u>CINC</u>	<u>TOF</u>	<u>MONTH</u>	<u>TEXTLINES</u>	<u>REPORTTYPE</u>	<u>CLASS</u>
3	051330	JUNE	214	DISUM	S
3	061445	JUNE	220	DISUM	S
3N	051440	JUNE	10	SITREP	T
3N	051543	JUNE	100	OPREP3	S
3N	051835	JUNE	20	SPIREP	C
3N	051930	JUNE	5	DISUM	T
3N	061550	JUNE	50	SITREP	S
3N	061600	JUNE	200	OPREP3	C
3N	061747	JUNE	30	SPIREP	S
3N	061820	JUNE	15	DISUM	C
3N	071430	JUNE	300	OPREP3	T
3N	071510	JUNE	40	SPIREP	S

3N	071545	JUNE	25	DISUM	T
3N	071650	JUNE	60	SITREP	S

The input file for this example contains statistics on teletype messages originated by several CINCs. Data fields include CINC, whose values are alphanumeric codes; TOF, whose values are the times of file for each message; REPORTTYPE, whose values include DISUM, SITREP, OPREP3, and SPIREP; CLASSIFICATION, whose values are codes C (Confidential), S (Secret), and T (Top Secret); and TEXTLINES, whose numeric values are the number of text lines per report type.

The RETRIEVE statement qualifies input file records containing CINC codes "3N" or "3 N" and message times of file between 0001 hours on June 6 and 2359 hours on June 8.

The arithmetic clause on the USE statement causes the number of lines of text to be summed together for each intersection of a row and column. If the clause was omitted the matrix value would simply give a count of how many documents had a particular classification.

A tabular report for the resulting cross tabulation is shown below:

	C	S	T
DISUM	15	35	25
OPREP3	200	0	700
SITREP	0	180	0
SPIREP	0	70	50

Example 5: EXPLICIT/USE Combination.

In this example, the explicit vector definition (EXPLICIT or *), in combination with USE, generates data counts for each of the unique data values occurring in the field specified by the USE option.

The following data is used:

<u>STATECODE</u>	<u>POPULATION</u>	<u>AREACODE</u>
AK	1000	A
PA	500	A
	500	C
NY	2000	B
AK	500	B
AK	600	C
AL	500	A
PA	1000	B
NY	1000	C
NJ	500	A
	1000	C
PA	5000	C

The objective of the report is to produce a tally of area codes by states showing both the numbers of small areas and a total distribution of areas by state.

The GIPSY command sequence is:

```
BUILD TABULAR REPORT.  
  ROWS. USE STATECODE.  
  COLUMNS. *  
    "SMALL AREAS" IF POPULATION LT 1000.  
    "TOTAL AREAS".  
END.
```

GIPSY dynamically assigns each unique occurrence of STATECODE to a row vector, creating row labels. The explicit statement creates the column labels. Each record is then effectively counted into the proper row and column of the report. As GIPSY proceeds sequentially through the file, it creates new rows (and resorts the row headers) as new values are encountered for STATECODE. If the value was previously encountered, it is already in the report; thus, GIPSY will find it and add one to the proper column in that row for each subsequent occurrence. The proper column is selected by inspecting the conditional expression in the explicit definition. Data will always qualify for the column TOTAL AREAS since it contains no conditional expression to select its data as does SMALL AREAS.

The tabular report resulting from the operations on the file contains a count for the single explicit vector, as follows:

	SMALL AREAS	TOTAL AREAS
(blank)	1	2
AK	2	3
AL	1	1
NJ	1	1
NY	0	2
PA	1	3

A selected population summary is produced by:

```
BUILD TABULAR REPORT. ROWS. *  
  ALASKA IF STATECODE EQ AK.  
  PENNA IF STATECODE EQ PA.  
  COLUMN. SELECT POPULATION.  
END.
```

The user file is the same as for the previous example. The tabular report for the above PCS is:

	POPULATION
ALASKA	2100
PENNA	6500

Example 6. SECTION Definition

The user's file contains the data values as follows:

FLIGHT NUMBER	NUMPASSENGER	DATE
FLT604	120	JUNE29
FLT725	100	JUNE29
FLT821	150	JUNE29
FLT604	200	JUNE30
FLT725	160	JUNE30
FLT821	180	JUNE30

The user wants to construct a report showing the number of passengers on each of the flights. Furthermore, the user wants to subdivide the flight numbers by the day they occurred. The request entered is:

```
FILE FLIGHTFILE
FDT FWTDT
BUILD TABULAR REPORT.
SECTIONS. USE DATE.
ROWS. USE FLIGHTNUMBER.
COLUMNS. SELECT NUMPASSENGER.
END.
RUN.
```

The statements specify a matrix with row labels taken from the contents of the FLIGHTNUMBER field of the records in the user's file. The section labels will be taken from the contents of the DATE field of the records in the user's file. The column label is NUMPASSENGER, which is the name of the specified field.

The tabular report for the resulting matrix is as follows:

JUNE29	NUMPASSENGER
FLT604	120
FLT725	100
FLT821	150

JUNE30	NUMPASSENGER
FLT604	200
FLT725	160
FLT821	180

Example 7. CATEGORY and SECTION

Let us assume a user data file of message traffic containing the data values represented as follows (one record per message):

CINC	TOF	MONTH	TEXTLINES	REPORTTYPE	CLASS
3	051330	JUNE	214	DISUM	S
3	061445	JUNE	220	DISUM	S
3N	051440	JUNE	10	SITREP	T
3N	051543	JUNE	100	OPREP3	S
3N	051835	JUNE	20	SPIREP	C
3N	051930	JUNE	5	DISUM	T
3N	061550	JUNE	50	SITREP	S
3N	061600	JUNE	200	OPREP3	C
3N	061747	JUNE	30	SPIREP	S
3N	061820	JULY	15	DISUM	C
3N	071430	JULY	300	OPREP3	T
3N	071510	JULY	40	SPIREP	S
3N	071545	JULY	25	DISUM	T
3N	071650	JULY	60	SITREP	S
3N	081350	JULY	70	SITREP	S
3N	081505	JULY	400	OPREP3	T
3N	081555	JULY	50	SPIREP	T
3N	081616	JULY	35	DISUM	S

The statements below will show the use of all four vector modes in a simple application using the data file shown above:

```

FILE MY/DATA/FILE
FDT MY/DATA/FILEFDT
BUILD TABULAR REPORT.
CATEGORY.  USE MONTH.
SECTION.   USE CLASS.
COLUMN.   USE (UNSORTED) CINC.
ROW.  RANGE TOF FROM 50000 TO 90000 BY 10000.
END.
RUN.
DISPLAY REPORT.

```

Since no arithmetic processes were implicitly or explicitly used in the tabular report definition, the system will build the tabular report using the count of occurrences of records broken out as specified in the BUILD TABULAR REPORT block. The reader should refer back to the file listing on the previous page to observe how the values were included on the tabular report.

	JUNE			JULY		
	3	3N	3 N	3	3N	3 N
C						
50000-59999	0	1	0	0	0	0
60000-69999	0	0	1	0	1	0
70000-79999	0	0	0	0	0	0
80000-89999	0	0	0	0	0	0
90000-90000	0	0	0	0	0	0

S	50000-59999	1	1	0	0	0	0
	60000-69999	1	2	0	0	0	0
	70000-79999	0	0	0	0	2	0
	80000-89999	0	0	0	1	2	0
	90000-90000	0	0	0	0	0	0
T	50000-59999	0	2	0	0	0	0
	60000-69999	0	0	0	0	0	0
	70000-79999	0	0	0	0	1	1
	80000-89999	0	0	0	0	1	1
	90000-90000	0	0	0	0	0	0

The above report shows the number of messages distributed over TOF and grouped by classification and organized by CINC code within each month. If a tally of number of lines of text were desired rather than number of messages simply add the statement CALCULATE DATA = TEXTLINES to the PCS.

3.3.2.2 Bar Graphs. The general form of the syntax for displaying a bargraph is:

```
DISPLAY BAR [GRAPH] [<y-axis title>] [<x-axis title>] [<value options>]
<graph details>.
```

The word GRAPH may be added, if desired, to improve readability of the statement. The <y-axis title> is composed as follows:

```
[OF] [Y-TITLE] "<literal>" [(SIZE <size option>, COLOR <color option>)]
```

The "<literal>" contains the text that will be displayed vertically along the y-axis to inform the viewer of the meaning of the numbers labeling the y-axis. The format for the <x-axis title> is:

```
X-TITLE "<literal>" [(SIZE <size option>,COLOR <color option>)]
```

The "<literal>" is the text that will be displayed horizontally between the x-axis labels and the legend which defines the bars on the x-axis.

The <size option> establishes the size of the text. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color of the text. If no color is specified, the current color is used.

The <value option> clause is used to label each individual bar with its numeric value. The syntax for this statement is:

```
[WITH] VALUE[S] { EXTERNAL
                   INTERNAL
                   BOTH } [(CENTER,SIZE <size option>, COLOR <color option>)]
```

The word WITH may be used to improve readability. The VALUE phrase causes the values to be added to the bargraph display. The system default is external labels. If INTERNAL is specified the value will be printed inside the top of each bar.

For stacked bargraphs, the EXTERNAL label provides a combined total at the top of each bar, while INTERNAL labels will provide the value of each stacked segment. If BOTH is requested, the resulting bargraph will contain individual values for each portion of the stacked bar plus a total value for the entire bar written at the top.

The CENTER option causes the INTERNAL labels to be vertically centered within each bar. The default will display the label inside the top of each bar.

The <size option> establishes the size of the label. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color of the label. If no color is specified, the current color is used.

The <graph details> define the rows, columns, categories and/or sections to be used to build the bargraph. They also define the format of the bargraph (e.g., grouped, stacked, etc.) shading and legend information. The various graph details are preceded by the verbs USING, STACKING, FOR or PAGING. All bar graphs must have either USING or STACKING specified. The FOR verb is optional, and is used to explicitly define the x-axis headers. The PAGING verb is used with reports containing sections or categories. Following is the syntax:

$\left\{ \begin{array}{l} \text{USING} \\ \text{STACKING} \\ [\text{FOR}] \\ \text{PAGING} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{ROWS} \\ \text{COLUMNS} \\ \text{SECTION} \\ \text{CATEGORIES} \end{array} \right\}$	$[\text{<vector name>["<legend name>"}]$ $[(\text{<shading/color options>})]$
--	--	--

The USING phrase defines the vector mode (ROW, COL, etc.) and the name of the vector(s) within that vector mode which identifies the set (or vector) of numbers to be plotted as bars. Referencing the limited tabular report last shown, we can produce the bargraphs in figure 3-7 with the command:

DISPLAY BAR GRAPH USING COLUMN 0-4.

Note that each value of column 0-4 is plotted with its corresponding ROW header identifying it on the x-axis.

Figure 3-8 was produced by issuing the command:

DISPLAY BAR USING ROW CCTC.

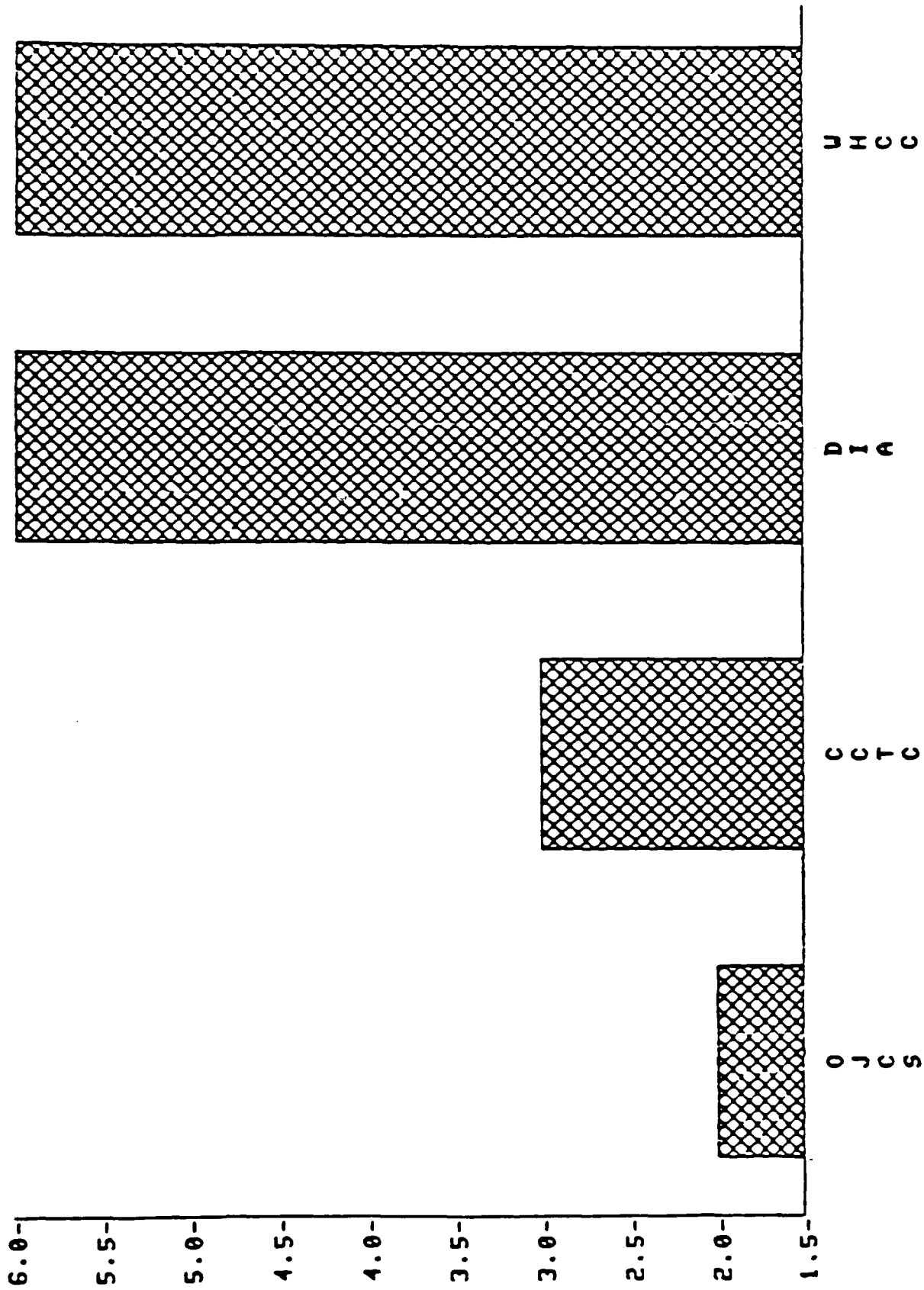


Figure 3-7. Bar Graph Using Specific Column

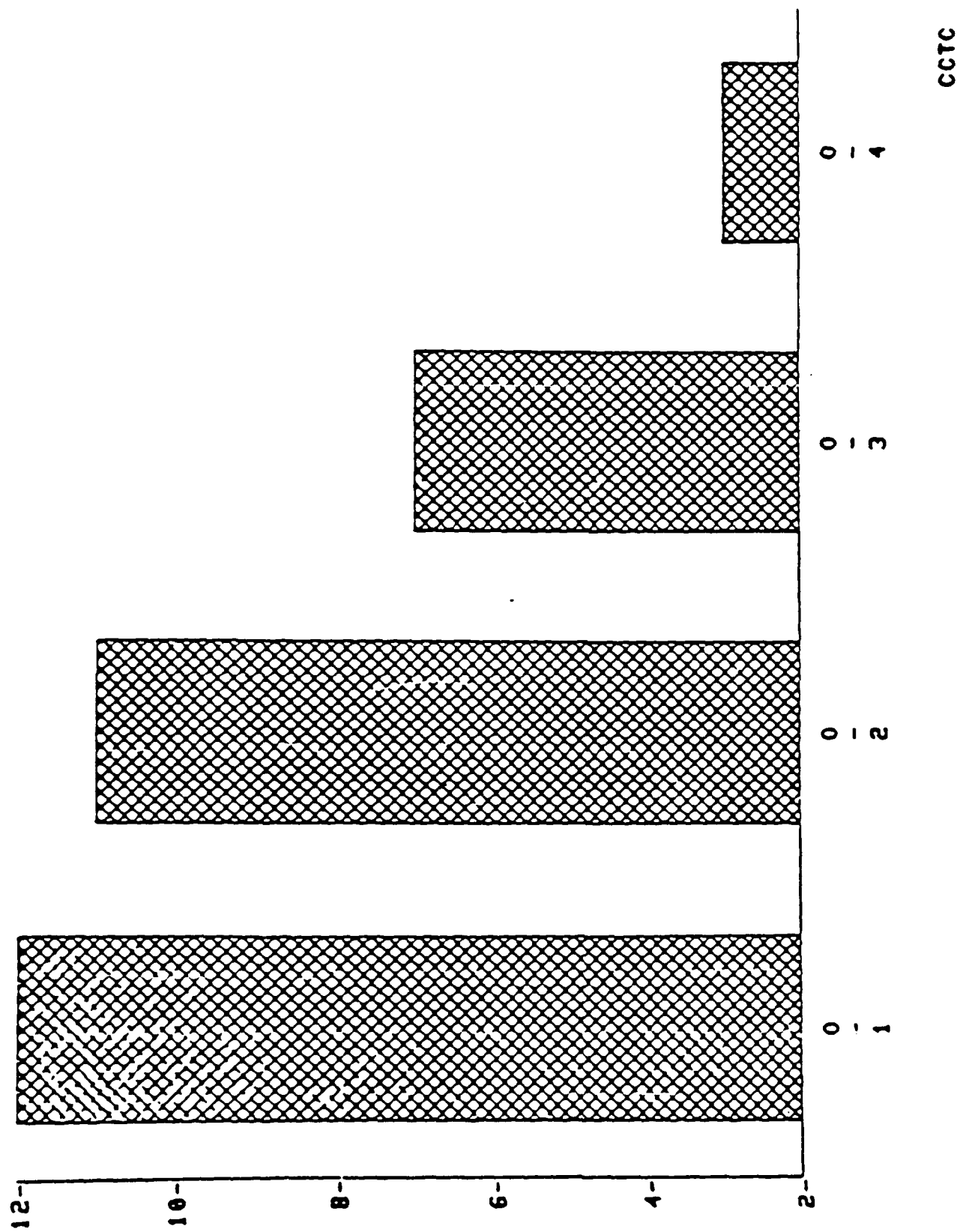


Figure 3-8. Bar Graph Using Specific Row

Note that GIPSY adjusted to the use of rows and plotted the column headers on the x-axis.

The set of parameters <vector name> ["<legend name>"] [(<shading/color options>)] may be repeated for each vector in the indicated vector mode. However, each repetition must be separated from the other by a comma. If more than one vector is supplied, they will be grouped with the corresponding vector mode producing what was referenced as a HISTOGRAM in earlier releases of GIPSY. This will be illustrated shortly.

We have used the <vector name> in our preceding examples; however, there are a few idiosyncrasies which must be pointed out. If the vector mode is numeric or contains special characters, the name of the vector mode must be enclosed in quotes. GIPSY will group together as many bars for as many vector names you specify. If more than one vector is used, GIPSY will automatically create a legend to identify the data elements. The name of the vector will be used unless a <legend name> is specified. The <legend name> must always be in quotes if used and it must immediately follow the <vector name> to which it applies.

GIPSY automatically determines the size of the bar and the spacing between the bars as a function of the number items to be plotted. It will also figure out shading patterns for the bars in order to separate the details. You may choose to override GIPSY's shade selection and apply your own. If you specify <shading/color options> for any item, GIPSY's shade selection mechanism is turned off; shading of all bars becomes your responsibility. The <shading/color options> on the DISPLAY BAR GRAPH command consists of shading line density (DENSITY), shading direction (SHADE), shading line color (COLOR), and the color fill for the bar (FILL). Recall from the syntax of the <shading/color options> that all of these options must be collected within parentheses following the vector name (or <legend name> if supplied). The first option describes the thickness of the shading line. This may be specified with one of three line densities:

DENSITY	{	LIGHT
		MEDIUM
		HEAVY
	}	

The second option describes the direction of the shading line and has the following syntax:

SHADE	{	LEFT
		RIGHT
		ACROSS
		UP
		DOWN
		NONE
	}	

LEFT refers to a left slant diagonal shading; RIGHT, refers to a right slant diagonal shading; ACROSS is horizontal shading; UP and DOWN are vertical shading. Since multiple directions may be specified for any item resulting in 16 different directional combinations, each direction should be separated from the other with a comma or the word AND.

For reports produced on a color graphic terminal, GIPSY provides the third capability of adding colors to the bar graph. The syntax for color options is:

```
{ COLOR }  
  FILL   } <color name>
```

The COLOR parameter defines which color the bar outline and shading will be drawn in. FILL defines the color with which the box will be filled. The <color name> is a color such as RED, GREEN, or BLUE defined in GIPSY for the particular terminal in use. See section 3.2.3.17 for further discussions of color usage.

The shading pattern and density for the bar graph should be carefully selected. They can convey significant implications, intended and unintended. They also have a significant impact on the appearance of the graph. There are a number of factors which must be considered in selecting these parameters including the number of bars to be displayed. In fact, it is possible to produce an ambiguous situation because the size of the bar does not permit the entire shading pattern to be obvious. The system default shading addresses many of these factors. However, there is no substitute for human selection based on the intent of the display.

Now we can specify the bar graph:

```
DISPLAY BAR GRAPH OF "NR. OF OFFICERS" USING COLUMNS  
  O-1 "ENSIGNS" (SHADE LEFT),  
  O-2 "LT/JG" (SHADE LEFT AND RIGHT),  
  O-3 "LIEUTENANT" (SHADE ACROSS, DOWN).
```

The display results are shown in figure 3-9. Note that we also specified a title that is appropriate for this graph:

```
TITLE "JUNIOR NAVAL OFFICERS IN JOINT SERVICE AGENCIES" (SIZE J);  
      "(WASHINGTON METRO AREA)" (SIZE MEDIUM).
```

The size of the characters used in most cases depends upon the current setting of the character size. In this case, the character size was set too large. If for some reason we wanted to have a different character size we could change it with the SET command, then reissue the display command. It would not be necessary to reissue the TITLE statement because the last title will remain in effect until changed by another TITLE statement.

The STACKING phrase which may also be used in the DISPLAY BAR GRAPH command allows you to graphically stack vectors and distinguish those vectors via

JUNIOR NAVAL OFFICERS IN JOINT SERVICE AGENCIES (WASHINGTON METRO AREA)

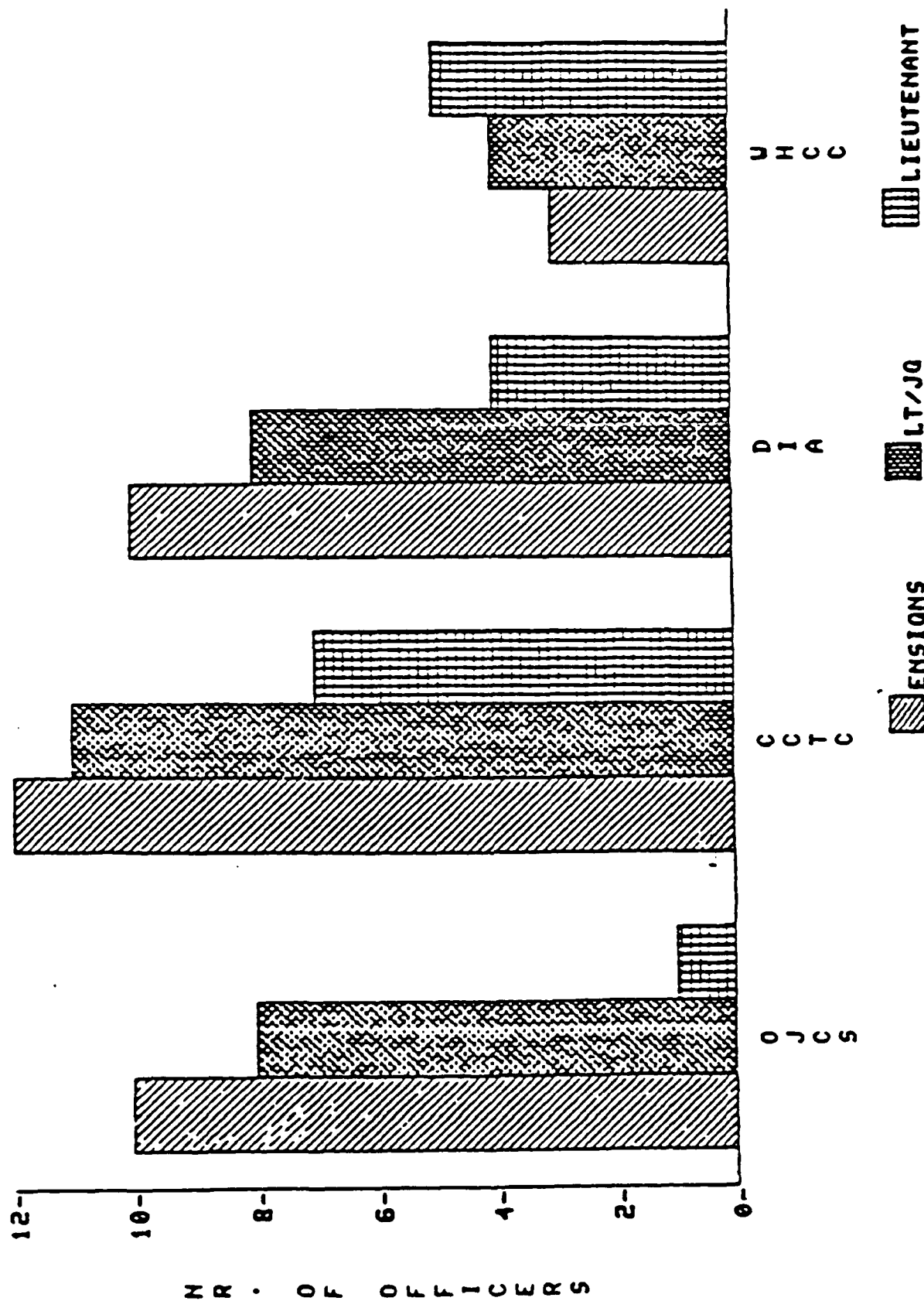


Figure 3-9. User Shading on Group Bar Graph

shading. Stacked vectors are plotted one on top of the other in the order specified. For example, figure 3-10 was produced with the command:

```
DISPLAY BAR STACKING ROWS
OJCS (SHADE ACROSS, UP, DENSITY HEAVY),
CCTC (SHADE ACROSS, AND UP, DENSITY MEDIUM),
WHCC (SHADE ACROSS AND UP, DENSITY LIGHT).
```

USING and STACKING are not mutually exclusive as it may first appear. You may use STACKING and USING in a single DISPLAY BAR command if your tabular report will support it. Your tabular report must have sections and/or categories in order to specify both a USING and a STACKING phrase. The resulting grouped stacked bar graph is a fairly complicated graph. The parenthetical options on the STACKING phrase will be used when both STACKING and USING appear in a single DISPLAY BAR GRAPH command. The legend will reflect the STACKING options and the USING vectors will be grouped and labelled on the x-axis as shown in figure 3-11, which was produced from our last tabular report using the following commands:

```
DISPLAY BAR USING CATEGORY ARMY, USAF;
STACKING COL 0-1 "2LT", 0-2 "1LT", 0-3 "CPT".
```

Pay particular attention to the placement of the information with respect to that specified in the command.

GIPSY provides the FOR phrase to override the default x-axis name, and to specify exactly which vector mode and which vector names will be used. The x-axis name normally defaults to all column headers if row-vectors are graphed, to all row headers if column-vectors are graphed.

The list of x-axis names takes any of the following forms or some combination thereof:

- (1) <vector name> [, <vector name>, ...]
- (2) <vector name a> THRU <vector name b>
- (3) <vector name b> THRU <vector name a>

In other words, you may spell out a list of specific vector names or a range of vector names, advancing forward or backward through the vector names. The vector names will be included on the graph in the order that they appear in the list.

If the FOR clause had been specified, the command for producing figure 3-10 could have been:

```
DISPLAY BAR USING CATEGORY ARMY, USAF;
STACKING COL 0-1 "2LT", 0-2 "1LT", 0-3 "CPT";
FOR ROWS OJCS, CCTC, WHCC.
```

JOINT SERVICE NAVAL OFFICER DISTRIBUTION (SELECTED WASHINGTON AREA AGENCIES)

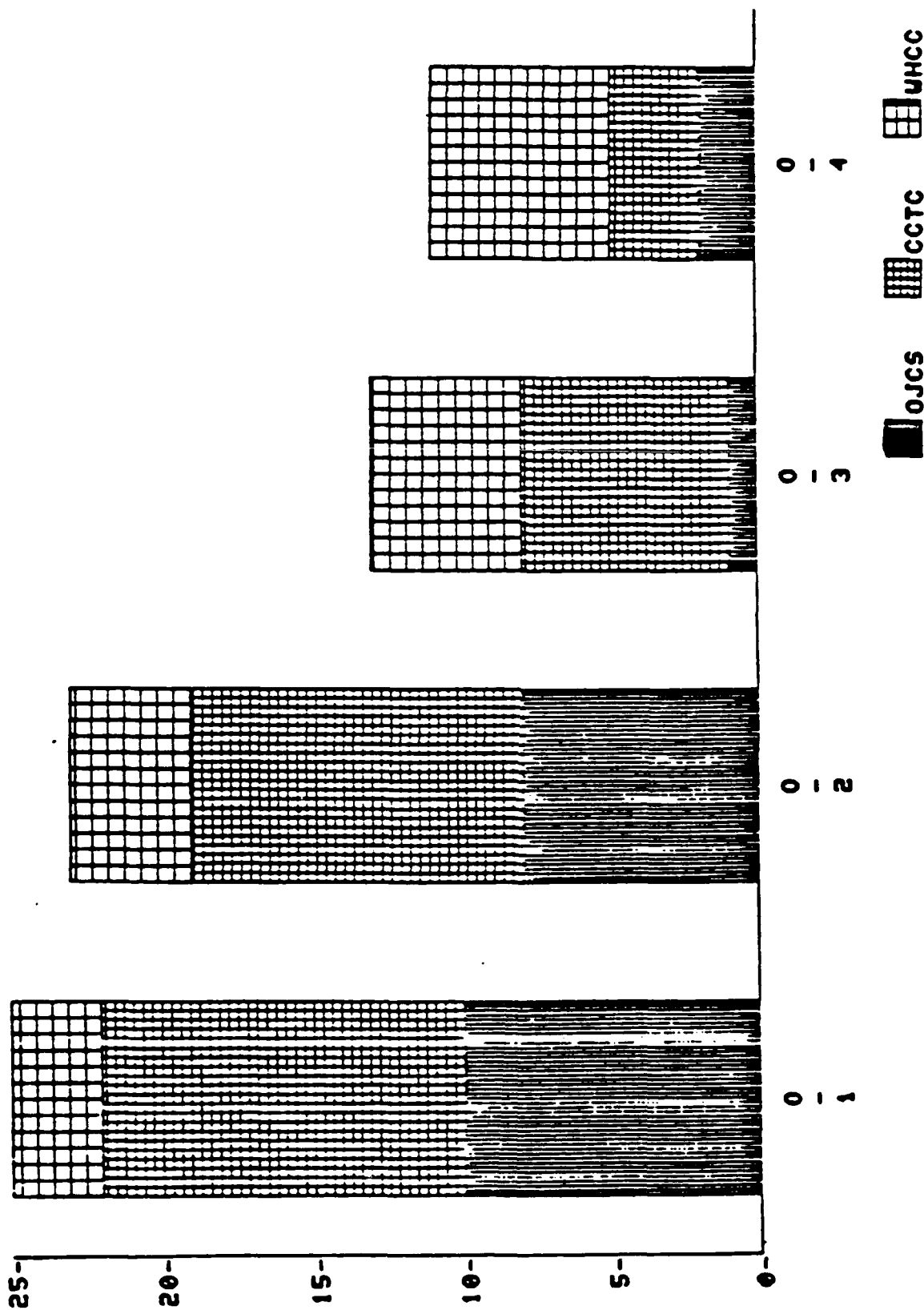


Figure 3-10. Stacked Bar Graph With Shading Densities

ARMY AND AIR FORCE OFFICERS (SELECTED WASHINGTON AREA AGENCIES)

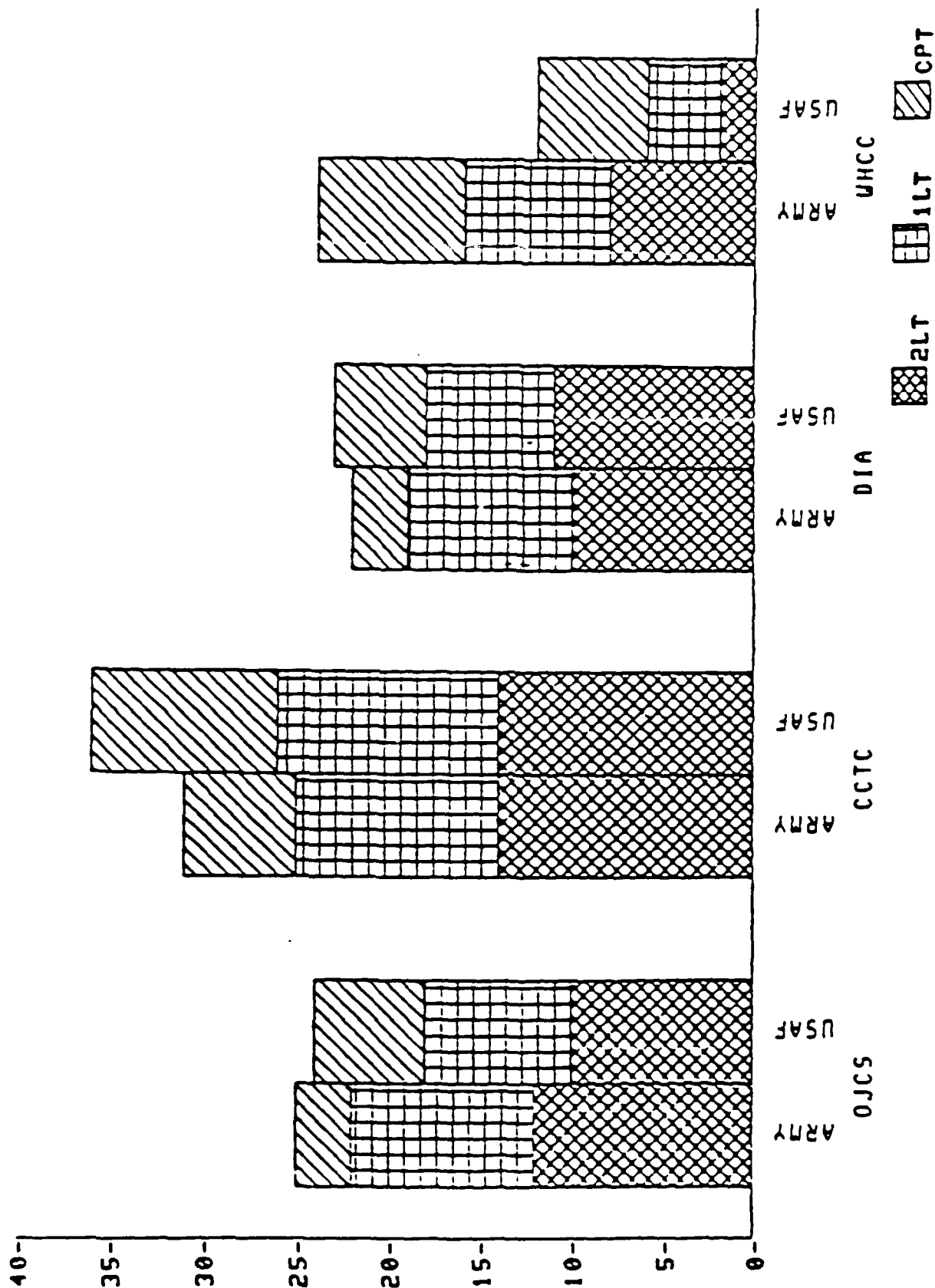


Figure 3-11. Stacking and Using Bar Graphs

The last phrase is PAGING. This phrase allows you to notify GIPSY to create multiple graphs from the single DISPLAY BAR GRAPH command; the next page will reflect the next parallel set of data, the next section, the next category, etc.

For each vector name listed, a new graph will be displayed on a new page using the same DISPLAY BAR GRAPH command. If no list is supplied, GIPSY will page on each vector in the referenced vector mode.

Returning to our growing DISPLAY BAR command, we could produce a graph for section FY80 (as page 1) and FY81 (as page 2) with a single command if we added a PAGING phrase.

```
DISPLAY BAR USING CATEGORY ARMY, USAF;  
    STACKING COL 0-1 "1LT", 0-2 "2LT", 0-3 "CPT";  
    FOR ROWS OJCS, CCTC, DIA, WHCC;  
    PAGING SECTIONS FY80, FY81.
```

When utilizing multiple phrases in a single DISPLAY BAR GRAPH command be aware that each phrase utilizes one of the vector modes of your report. You cannot page on the same vector mode that you are STACKING; you cannot stack on the same mode that you are USING for your x-axis, etc.

All our examples thus far have used row or column as the actual vector graphed. But, the reader should particularly note that any mode can be used for any phrase. Observe the result of this command in figure 3-12. We're using the tabular report without the LIMIT.

```
D B U SEC; PAGE ROWS; S COL; FOR CAT.
```

Note the abbreviations (D for DISPLAY, B for BAR, U for USING, S for Stacking, ...)

3.3.2.3 Histograms. A histogram is simply a bar graph with no spacing between the individual bars. It is ideal for displaying trends of data or sequences of information. The syntax is the same as that for the bar graph except that the word HISTOGRAM is substituted for BAR:

```
DISPLAY HISTOGRAM [<y-axis title>] [<x-axis title>] [<value options>]  
<graph details>.
```

The report in figure 3-13 was produced by the command:

```
DISPLAY HISTOGRAM OF "NUMBER OF HOURS" USING COLUMN JANUARY.
```

3.3.2.4 Point and Line Graphs. The syntax and semantics for the point graph and the line graph are very closely related to the bar graph discussed in section 3.3.2.2. The general form of the syntax is:

ARMY AND AIR FORCE OFFICERS
(SELECTED WASHINGTON AREA AGENCIES)
OJCS

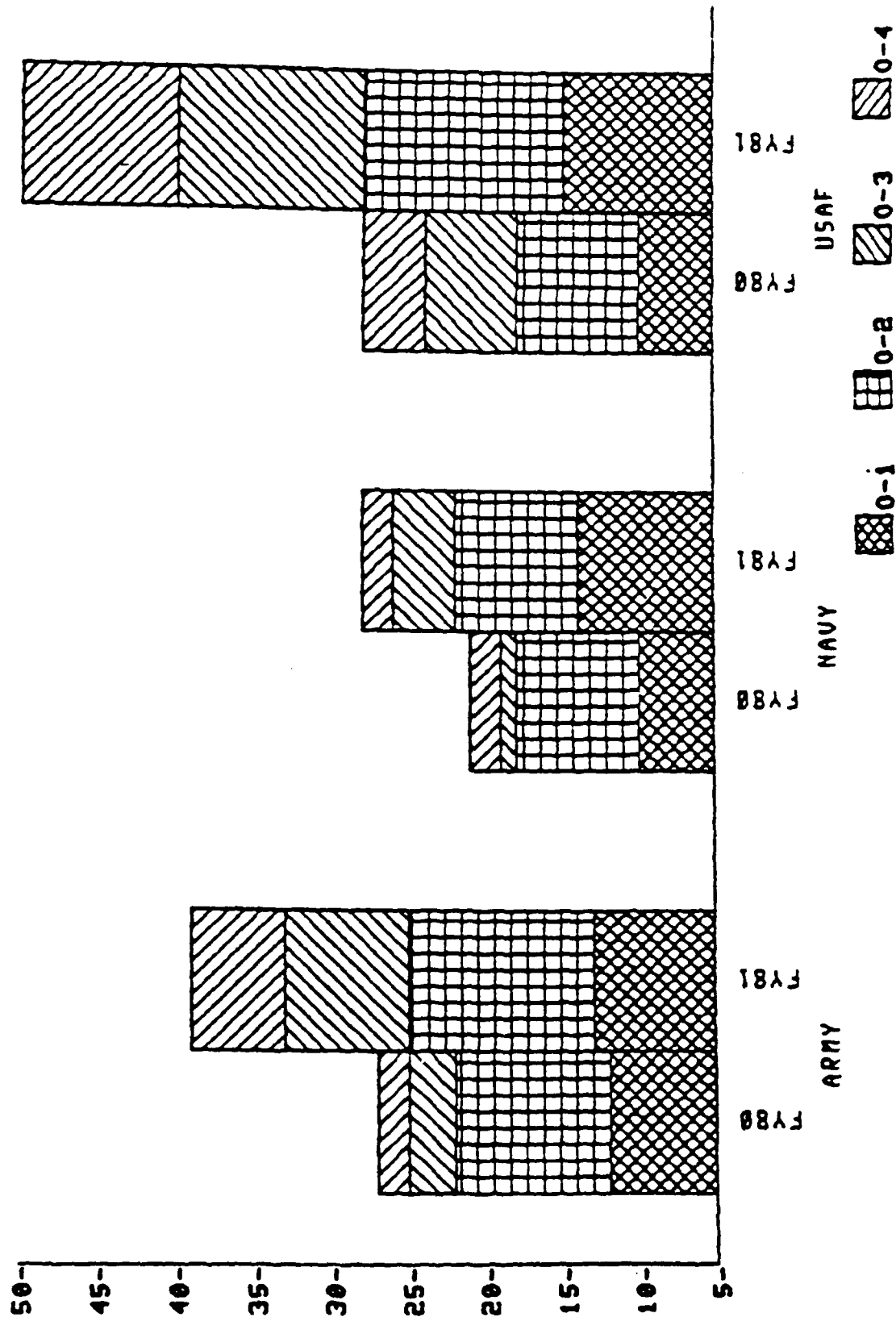


Figure 3-12. Paged Bar Graph (Part 1 of 4)

ARMY AND AIR FORCE OFFICERS (SELECTED WASHINGTON AREA AGENCIES) CCTC

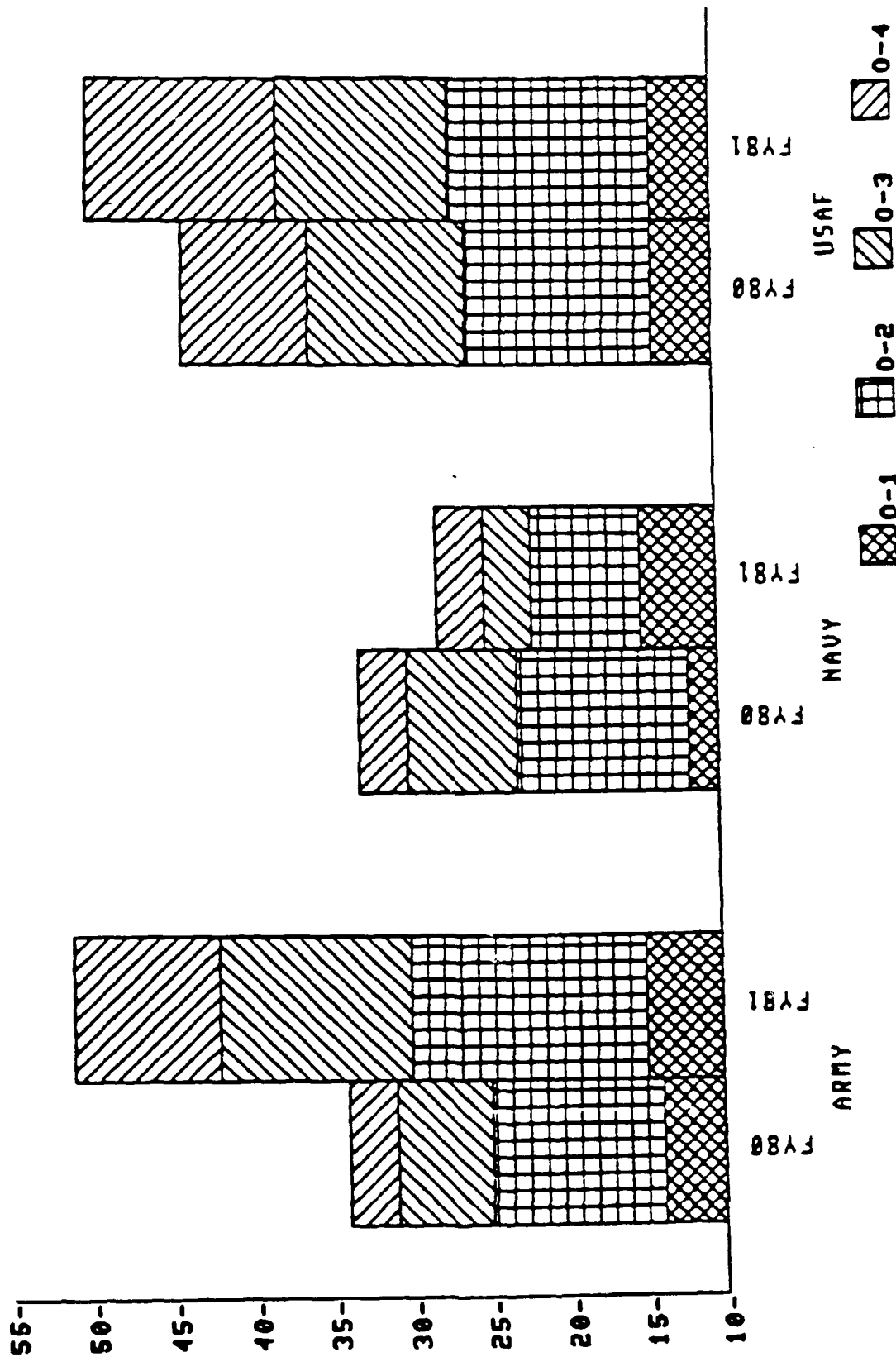


Figure 3-12. Paged Bar Graph (Part 2 of 4)

ARMY AND AIR FORCE OFFICERS
(SELECTED WASHINGTON AREA AGENCIES)
DIA

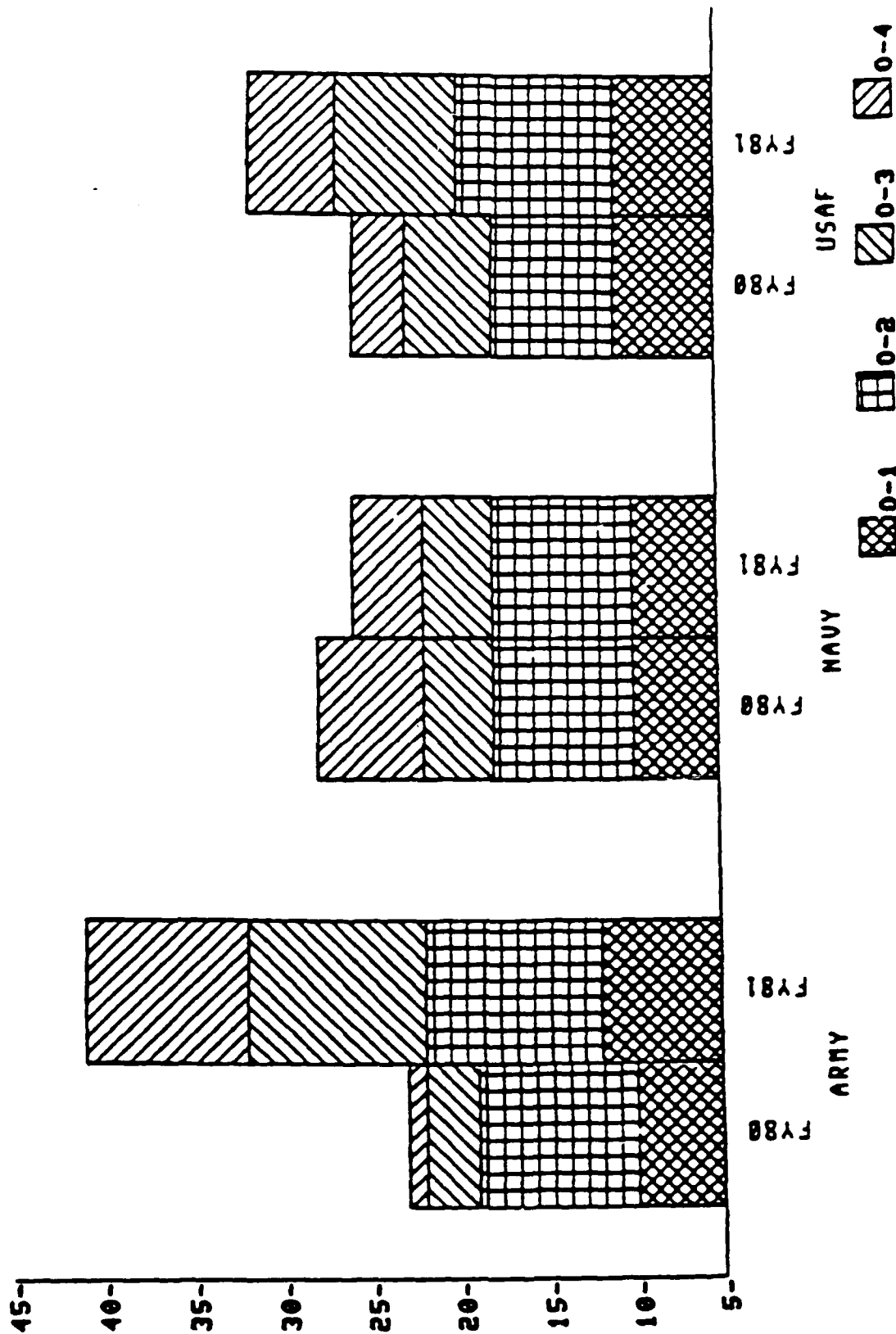


Figure 3-12. Paged Bar Graph (Part 3 of 4)

ARMY AND AIR FORCE OFFICERS (SELECTED WASHINGTON AREA AGENCIES) UHCC

4 OF 4

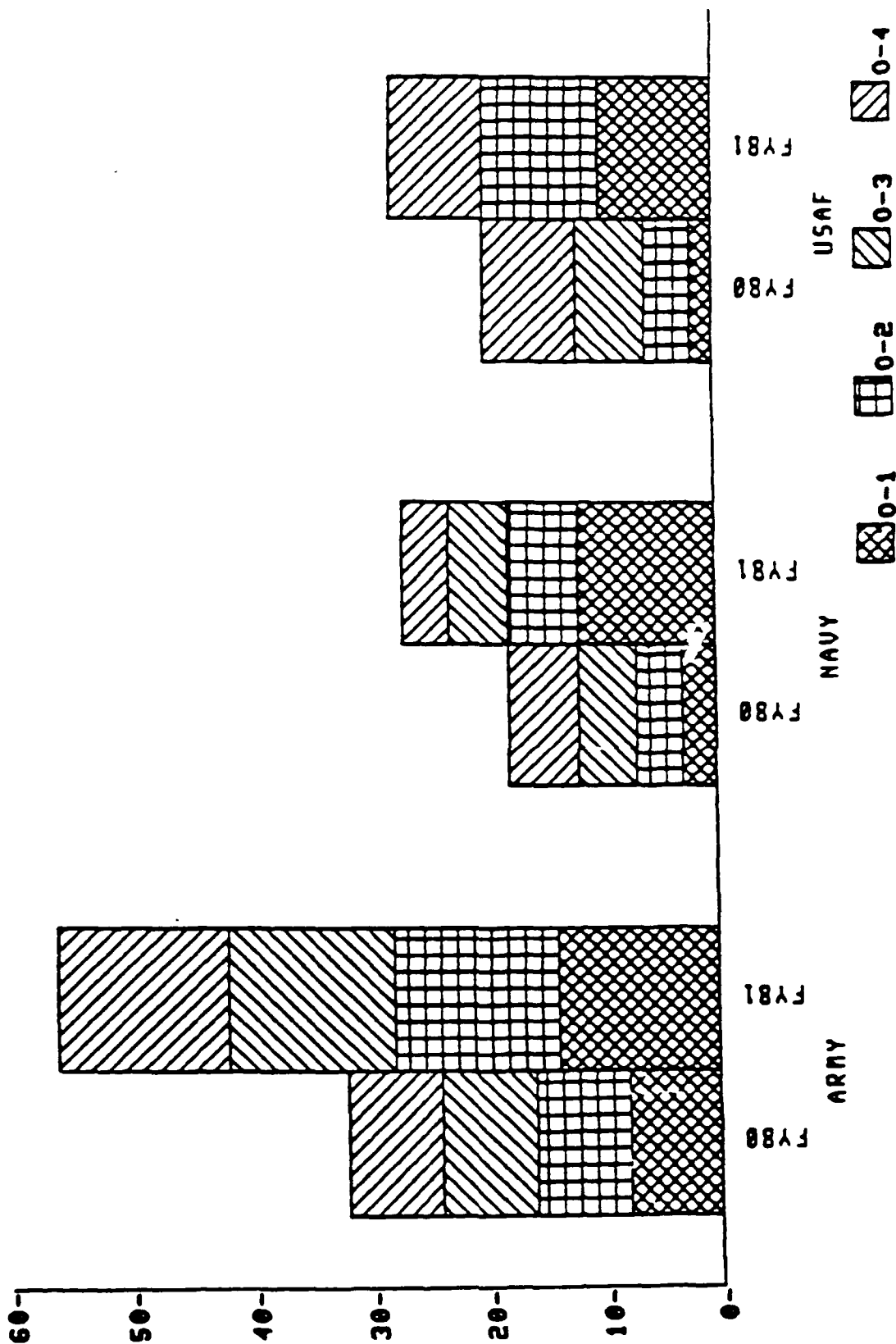


Figure 3-12. Paged Bar Graph (Part 4 of 4)

HOURS OF SUNSHINE DURING EACH DAY MONTH OF JANUARY



Figure 3-13. Sample Histogram

```

DISPLAY { POINT } [GRAPH] [<y-axis title>] [<x-axis title>] [<value
        LINE   option>] <graph details>.

```

The point graph is a proper subset of the line graph. Consequently we will discuss the line graph and indicate the limitations for the point graph. The word GRAPH may be added, if desired, to improve readability of the statement. The <y-axis title> is composed as follows:

```
[OF] [Y-TITLE] "<literal>" [(SIZE <size option>, COLOR <color option>)]
```

The "<literal>" contains the text that will be displayed vertically along the y-axis to inform the viewer of the meaning of the numbers labeling the y-axis.

The format for the x-axis title is:

```
X-TITLE "<literal>" [SIZE <size option>, COLOR <color option>]
```

The "<literal>" is the text that will be displayed horizontally between the x-axis labels and the legend which defines the points or lines.

The <size option> establishes the character size of the text for the titles. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color to be used for the text of the titles. If no color is specified the current color is used.

The <value options> has the format.

```
[WITH] VALUE[S] [(SIZE <size option>, COLOR <color option>)]
```

This phrase will cause the actual value of each point on the graph to be plotted as a label next to that point.

The word WITH may be added, if desired, to improve readability of the statement.

The <size option> establishes the size of the data to be used as the label. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color of the data to be used with the label. If no color is specified the current color is used.

The <graph details> on the DISPLAY LINE GRAPH and DISPLAY POINT GRAPH command, like the DISPLAY BAR GRAPH, may be made up of USING, STACKING, FOR or PAGING phrases. The syntax and semantics are the same as DISPLAY BAR GRAPH except

that the combined USING and STACKING is undefined and the graph details refer to line types and points rather than shading and density.

The DISPLAY LINE GRAPH and DISPLAY POINT GRAPH command may contain either a USING or a STACKING phrase, specifically:

$$\left\{ \begin{array}{l} \text{USING} \\ \text{STACKING} \end{array} \right\} \left\{ \begin{array}{l} \text{ROW} \\ \text{COLUMN} \\ \text{SECTION} \\ \text{CATEGORY} \end{array} \right\} [\text{<y-vector> } ["\text{<legend name>}"] [(\text{<line/point options>})]]$$

USING will cause each vector to be plotted against its own value; STACKING will cause each point on each succeeding line to be the sum of all corresponding points on all previous lines. The <y-vector> is the vector in the tabular report containing the y values of the line or point. The x values will be the vector headers corresponding to each of the y values.

For example, if columns are used for the y-axis, the row headers will be plotted as labels on the x-axis. There is no predefined proportion between the x value and the y value as in the result of some equation $y=f(x)$. The <legend name> specifies the name to be used in the legend rather than the vector header from the tabular report; if a <legend name> is not specified the vector header will be used.

The <line/point options> are used to distinguish differences among the lines on the graph. They consist of a line type, a point, a point size, and a color. The specific syntax for the <line/point option> on the DISPLAY LINE or DISPLAY POINT command is:

LINE -	$\left\{ \begin{array}{l} \text{SOLID} \\ \text{DOTTED} \\ \text{DASHED} \\ \text{DOT-DASH} \end{array} \right\}$
POINT -	" single character "
SIZE -	$\left\{ \begin{array}{l} \text{JUMBO} \\ \text{LARGE} \\ \text{MEDIUM} \\ \text{SMALL} \end{array} \right\}$
COLOR -	<color name>

You may use any one or a combination of these options as long as each option is separated from the other with a comma. The single character on the POINT option will be plotted on the line at the point representing the value of the line at that point. The size of the plotted character will be the current character size setting unless a SIZE option is included, in which case the specified size will be used. The LINE option is used only for the DISPLAY LINE command.

The capability to display a colored line may be done one of two ways. The first requires the user to specify the in-line COLOR options in the DISPLAY LINE [GRAPH] command which identifies an individual color for a specific line. The option to allow GIPSY to automatically select an individual color for every line may be accomplished by using the following command:

SET LINE COLOR { OFF
ON } .

The default is SET LINE COLOR OFF. If SET LINE COLOR ON is issued, all subsequent lines will be colored unless the in-line COLOR options or SET LINE COLOR OFF option is used. Colors will be obtained from the default color table when the SET LINE COLOR ON option is used.

GIPSY will select default points to identify the various lines on the graph if no <line/point options> are specified.

This statement may also have a FOR and a PAGING phrase which serves the same function as described for the bar graph in section 3.3.2.2.

A typical line graph as shown in figure 3-14 was generated with the command:

DISPLAY LINE GRAPH USING CATEGORY ARMY, NAVY; FOR COLUMNS
O-1 THRU O-4.

3.3.2.5 The Curve Graph. The curve graph is the classic graph of the variables from an equation of the form $y=f(x)$. In this form there is a numeric relationship between the x-axis and the y-axis. This relationship does not exist in GIPSY's line graph. The curve graph typically requires a little more advanced planning because the x-vector and its corresponding y-vector are extracted from the body of the report. The vector names are only used as identifiers for the individual vectors. The actual value of an x-axis label has no numeric value on a line graph. The curve graph differs in that the x-axis labels are numeric values. The graph assumes a rectangular Cartesian coordinate system with a definite numeric relationship between the x-axis scaling and y-axis scaling. The general form of the syntax for the DISPLAY CURVE command is:

DISPLAY CURVE [GRAPH] [<y-axis title>] [<x-axis title>] <graph details>.

The word GRAPH may be added to improve readability. The <y-axis title> is composed as follows:

[OF] [Y-TITLE] "<literal>" [(SIZE <size option>, COLOR <color option>)]

The "<literal>" contains the text that will be displayed vertically along the y-axis to inform the viewer of the meaning of the numbers labeling the y-axis. The format for the <x-axis title> is:

OFFICERS IN JOINT CHIEFS OF STAFF AGENCIES (WASHINGTON METRO AREA)

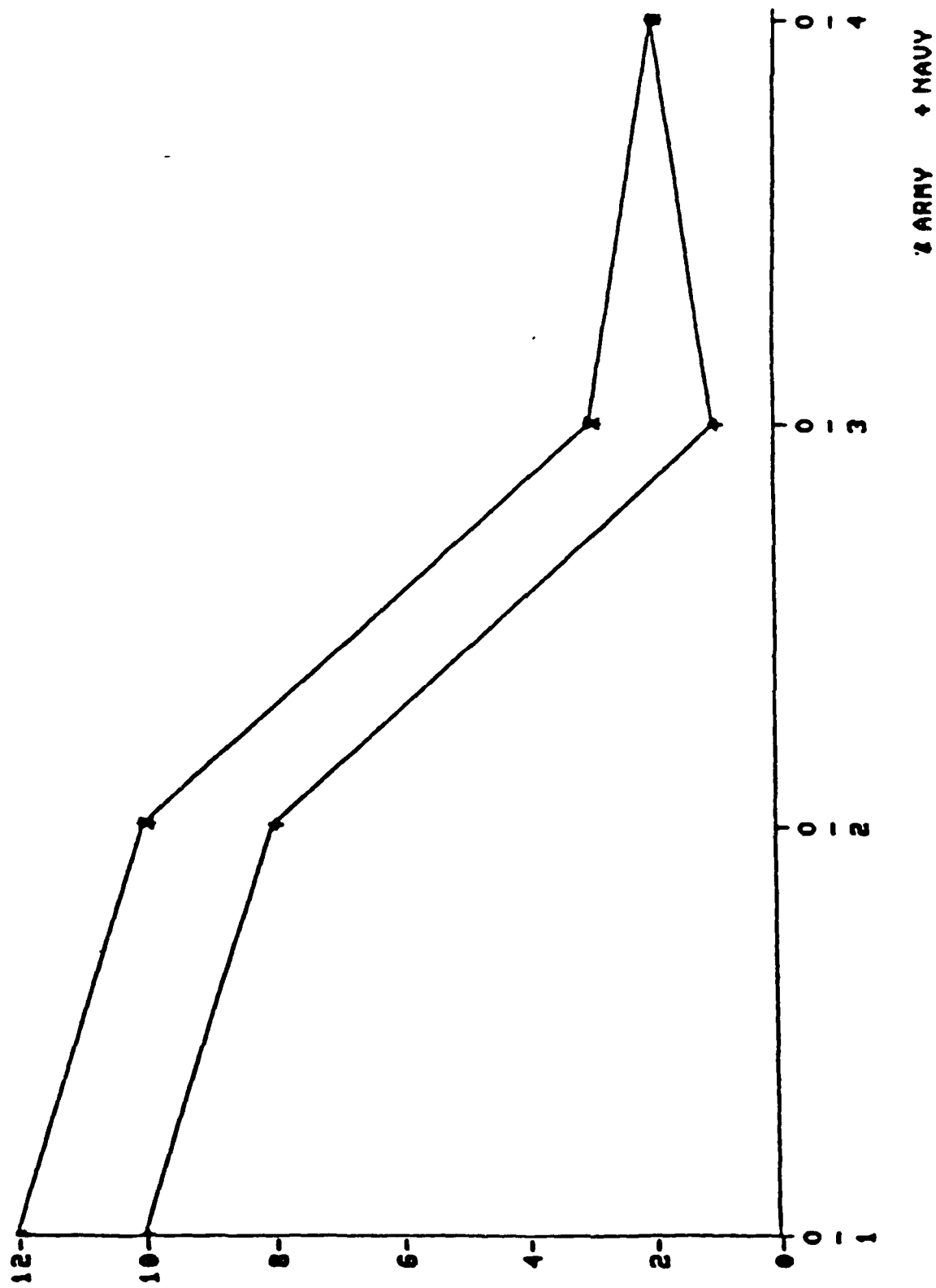


Figure 3-14. Line Graph With Using Phrase

X-TITLE "<literal>" [(SIZE <size option>, COLOR <color option>)]

The "<literal>" is the text that will be displayed horizontally between the x-axis labels and the legend which defines the lines being displayed.

The <size option> establishes the size of the text. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color of the text. If no color is specified, the current color is used.

The <graph details> define the vector mode of data from the tabular report to be used to plot the curve graph. The various <graph details> are preceded by the verbs USING, FOR or PAGING. All curve graphs must have USING specified to identify the vector mode and all the sets of x-y vector pairs to be plotted. The FOR and PAGING are optional.

The following syntax should be used to describe the <graph details> :

```
USING <vector mode>, <curve parameter>
[;FOR <vector mode> <list of identifiers to be included>]
[;PAGING <vector mode> <list of vectors for pages>] .
```

The USING phrase defines the <vector mode> and <vector name> which identifies the set of numbers to be plotted. The <vector mode> is the usual ROWS, COLUMN, SECTION or CATEGORY. This <vector mode> is used to tell GIPSY which vector mode of data from the tabular report will be used for the abscissa (x) and ordinate (y) values. The <curve parameter> details the actual name of the abscissa and ordinate vectors, the type of line to be used in plotting the curve, point markers, and color to be used. The <curve parameter> is made up of one or more groupings of abscissa and ordinate descriptors sets; the sets must be separated from each other by a semicolon; each vector within the set must be separated from the other by a comma. A single set is of the form:

```
X=<x-vector name>, Y=<y-vector name> ["<legend name>"] [<line/point options>]
[,<y-vector name 2> ["<legend name>"] [<line/point options>]] . .
[,<y-vector name n> ["<legend name>"] [<line/point options>]].
```

This form will create a family of curves all sharing the same x values.

This X,Y pair definition may be repeated as often as needed; each repetition must be separated from the other by a semicolon.

Let us try to clarify this with an example from the tabular report in figure 3-15. Note the use of the language in the command:

```
DISPLAY CURVE USING COLUMNS X=DAY
Y=LOSSES (LINE DOTTED), AVAILABLE (LINE DASHED).
```


REPORT ON LOSSES FOR FIRST 100 DAYS

1 OF 1

	DAY	LOSSES	AVAILABLE
01	0	5.0	70.0
02	2	5.0	69.0
03	4	5.0	68.0
04	6	5.0	66.5
05	8	5.3	65.0
06	10	5.8	63.0
07	12	6.0	61.0
08	14	6.5	59.0
09	16	7.0	56.5
10	18	7.8	54.0
11	20	8.5	51.0
12	22	9.3	48.0
13	24	10.5	44.0
14	26	12.0	40.0
15	28	13.3	36.0
16	30	15.0	32.0
17	32	16.5	28.5
18	34	19.0	26.0
19	36	22.0	24.0
20	38	25.5	22.0
21	40	29.0	20.0
22	42	34.0	18.5
23	44	39.0	17.0
24	46	44.0	15.8
25	48	50.0	15.0
26	50	56.0	14.0
27	52	59.0	13.0
28	54	61.0	12.5
29	56	60.0	12.0
30	58	49.0	11.8
31	60	35.5	11.3
32	62	30.8	11.0
33	64	27.5	10.5
34	66	25.3	10.3
35	68	23.5	10.0
36	70	22.0	9.8
37	72	21.0	9.8
38	74	20.0	9.7
39	76	19.0	9.5
40	78	18.0	9.5
41	80	16.8	9.4
42	82	15.8	9.3
43	84	14.8	9.2
44	86	13.0	9.1
45	88	12.8	9.0
46	90	12.0	9.0
47	92	11.3	9.0
48	94	11.0	9.0
49	96	10.5	9.0
50	98	10.0	9.0

Figure 3-15. Tabular Report for Curve Examples

The results are shown in figure 3-16. In this case, we used the columns as the vector mode and GIPSY automatically selected the rows to determine the sequence of points to plot with the default of plotting all points. The FOR phrase allows you to specify which items within the specified x vector are to be plotted. If the report has categories or sections it allows you to select the vector mode to be used.

3.3.2.6 The Step Graph. The step graph, like the curve graph, assumes a numeric relationship between the x-axis and the y-axis. For a given set of x-axis values, one or more sets of y-axis values can be plotted, each set of y-values being a function of the x-values. Unlike the curve graph, however, it does not represent a continuous relationship but shows an instantaneous change in y-value at each distinct x-value. The y-values between the distinct x-values are represented as a horizontal line having the value of the previous y, thus giving "steps" up or down the y-axis. Figure 3-17 is an example of such a step graph. The step graph is a good tool for showing distinct changes of a quantity over a period of time, such as shipments of supplies which arrive on a periodic basis. The general form of the syntax for the DISPLAY STEP command is:

```
DISPLAY STEP [GRAPH] [<y-axis title>] [<x-axis title>] <graph details>.
```

The word GRAPH may be added to improve readability. The <y-axis title> is composed as follows:

```
[OF] [Y-TITLE] "<literal>" [(SIZE <size option>, COLOR <color option>)]
```

The "<literal>" contains the text that will be displayed vertically along the y-axis to inform the viewer of the meaning of the numbers labeling the y-axis.

The format for the x-axis title is:

```
X-TITLE "<literal>" [(SIZE <size option>, COLOR <color option>)]
```

The "<literal>" is the text that will be displayed horizontally between the x-axis labels and the legend which defines the lines being displayed.

The <size option> establishes the size of the text. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color of the text. If no color is specified, the current color is used.

The <graph details> define the vector mode to be used to build the step graph. The various <graph details> are preceded by the verbs USING, SHOW, FOR, and PAGING. All step graphs must have USING specified. The SHOW verb is optional and allows the user to show the relationship between two step functions by shading or color filling between the lines representing the functions. The FOR verb is also optional and is used to explicitly define the x-axis names. The PAGING verb is used with reports containing sections or categories.

REPORT ON LOSSES FOR FIRST 100 DAYS

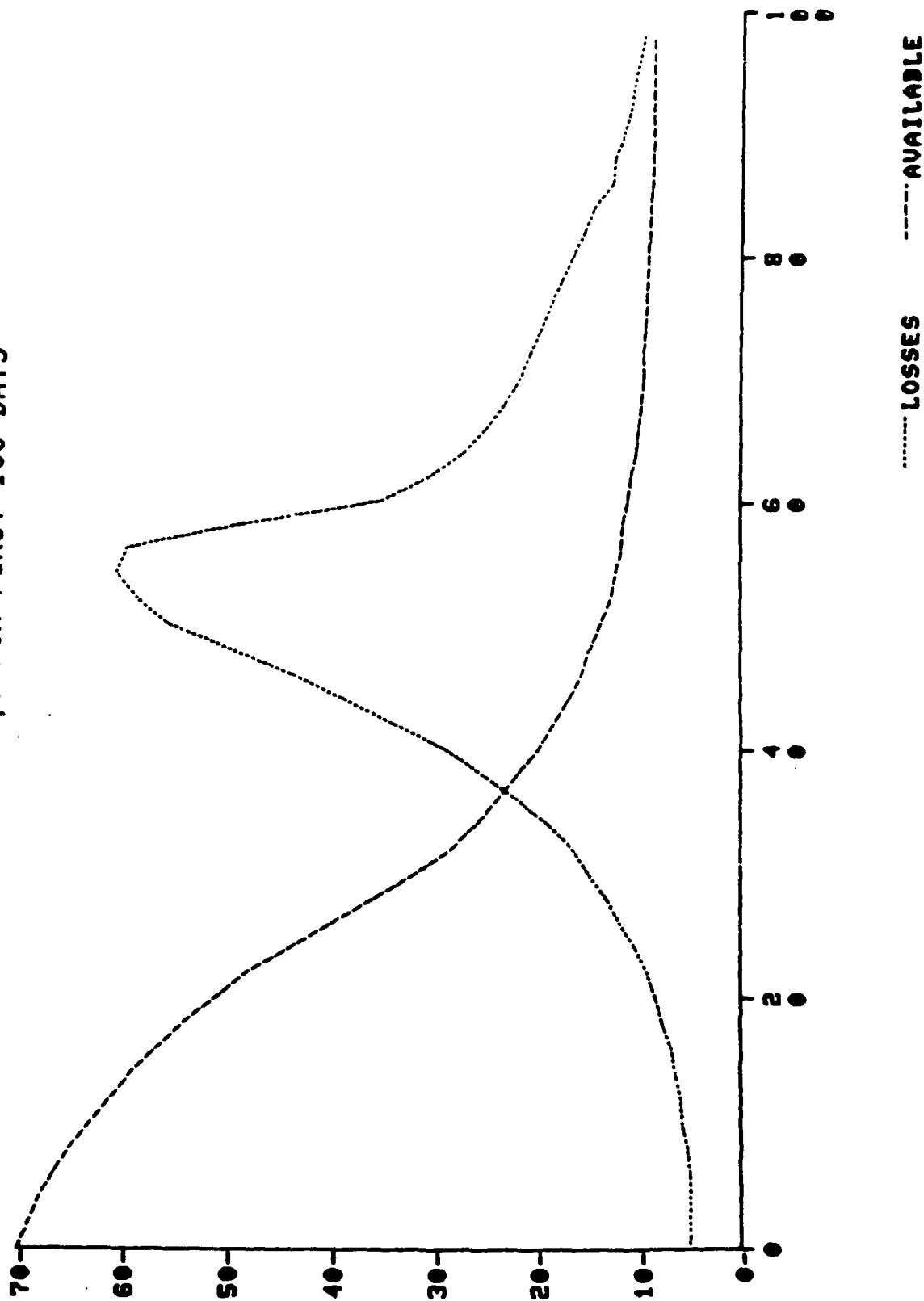


Figure 3-16. Sample Curve Display

PROJECTED MATERIAL ON HAND BY DAY

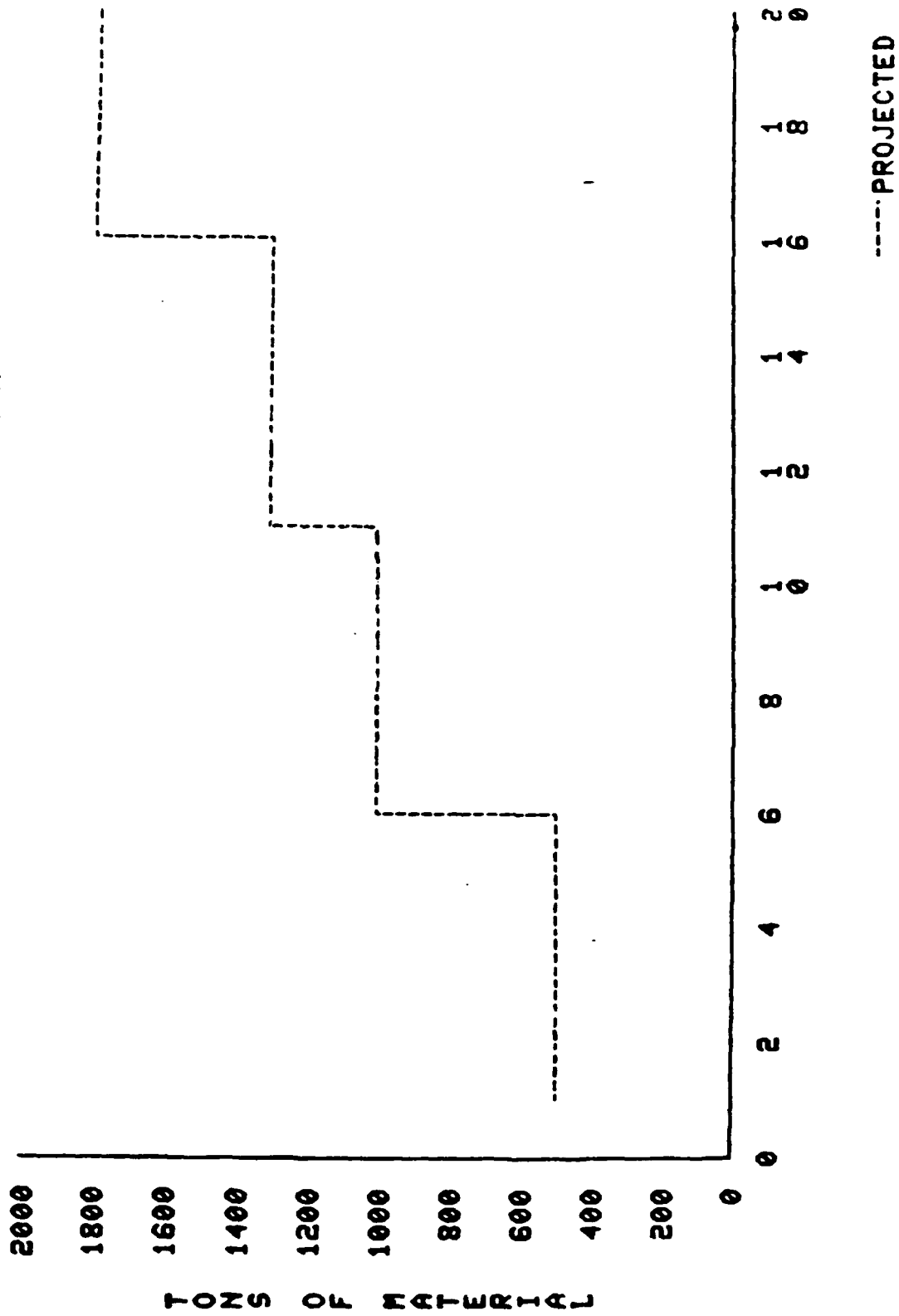


Figure 3-17. Sample Step Graph

Following is the syntax for the above mentioned verbs:

```

USING <vector mode> X=<vector name>,Y=<vector name> "<legend name>"
    [(<line options>)][,<vector name> "<legend name>" [(<line
                                options>)]] [, . . .] ;

[SHOW <vector name a> { LT
                       <
                       GT
                       > } <vector name b>["<legend>"][(<shade/fill
                                options>)]];

[FOR <vector mode> [<vector-name list>]];

[PAGING <vector mode> [<vector list>]].

```

The USING phrase defines the <vector mode> and the <vector names> which identify the set of numbers to be plotted. The <line options> may be specified as:

```

line option - [ COLOR <color option>
               LINE <line type> ]

```

The <color option> identifies the color of the line. If no color is specified, the current color is used.

The <line option> is used to distinguish differences among the lines on the graph.

The specified syntax for the <line type> is:

```

LINE { SOLID
      DOTTED
      DASHED
      DOT-DASH }

```

The SHOW option allows the user to show the relationship between two step functions by shading or color filling between the lines representing the functions.

The FOR and PAGING phrases are optional. The FOR verb is optional and is used to explicitly define the x-axis labels. The PAGING verb is used with reports containing sections or categories. The following is the syntax:

```
{ FOR
  PAGING } <vector mode> [<vector name> ["<legend name>"] [(  

  <shading/color options>)]
```

The following examples are provided to clarify the use of the STEP graph. Figure 3-18 shows a tabular report which will be used in producing the following STEP graphs. Figure 3-17 shows the results of the command:

```
D STEP OF "TONS OF MATERIAL" USING COLUMNS X=DAY, Y=PROJECTED (LINE
DASHED).
```

Figure 3-19 illustrates the use of the SHOW option to highlight where the PROJECTED and ACTUAL values differ. The command issued was:

```
D STEP OF "TONS OF MATERIAL" USING COLS X=DAY, Y=PROJECTED, ACTUAL; SHOW
ACTUAL GT PROJECTED; SHOW ACTUAL LT PROJECTED.
```

3.3.2.7 Gantt Chart. The gantt chart is a time-series chart used to depict one or more activities for a specific event. Used primarily in logistics planning to graphically depict deployment of men, units, or material, gantt charts can also be used as a management tool to show production schedules.

The general form of the syntax for the DISPLAY GANTT command is:

```
DISPLAY GANTT [CHART] [<y-axis title>] [<x-axis title>] <chart details> .
```

The word CHART may be added to improve readability. The <y-axis title> is composed as follows:

```
[OF] [Y-TITLE] " <literal> " [(SIZE <size option> ,COLOR <color option> )]
```

The "<literal>" contains the text that will be displayed vertically along the y-axis to inform the viewer of the meaning of the activities labeling the y-axis. The format for the <x-axis title> is:

```
X-TITLE "<literal>" [(SIZE <size option> ,COLOR <color option> )]
```

The "<literal>" is the text that will be displayed horizontally between the x-axis labels and the legend which defines the activities being displayed.

The <size option> establishes the size of the text. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

The <color option> identifies the color of the text. If no color is specified, the current color is used.

The <chart details> define the vector mode to be used to build the gantt chart. The various <chart options> are preceded by the verbs USING, PLANNED, ACTUAL, REQUIRED, FOR, and PAGING. The syntax for the use of these verbs is:

PROJECTED VRS. ACTUAL TONS OF MATERIAL ON HAND BY DAY

	DAY	PROJECTED	ACTUAL
DAY-1	1	500	0
DAY-2	2	500	0
DAY-3	3	500	700
DAY-4	4	500	700
DAY-5	5	500	700
DAY-6	6	1000	900
DAY-7	7	1000	900
DAY-8	8	1000	1200
DAY-9	9	1000	1300
DAY-10	10	1000	1300
DAY-11	11	1300	1300
DAY-12	12	1300	1600
DAY-13	13	1300	1600
DAY-14	14	1300	1700
DAY-15	15	1300	1700
DAY-16	16	1800	1700
DAY-17	17	1800	1700
DAY-18	18	1800	1700
DAY-19	19	1800	1700
DAY-20	20	1800	1700

Figure 3-18. Step Graph Data

PROJECTED VRS. ACTUAL MATERIAL ON HAND BY DAY

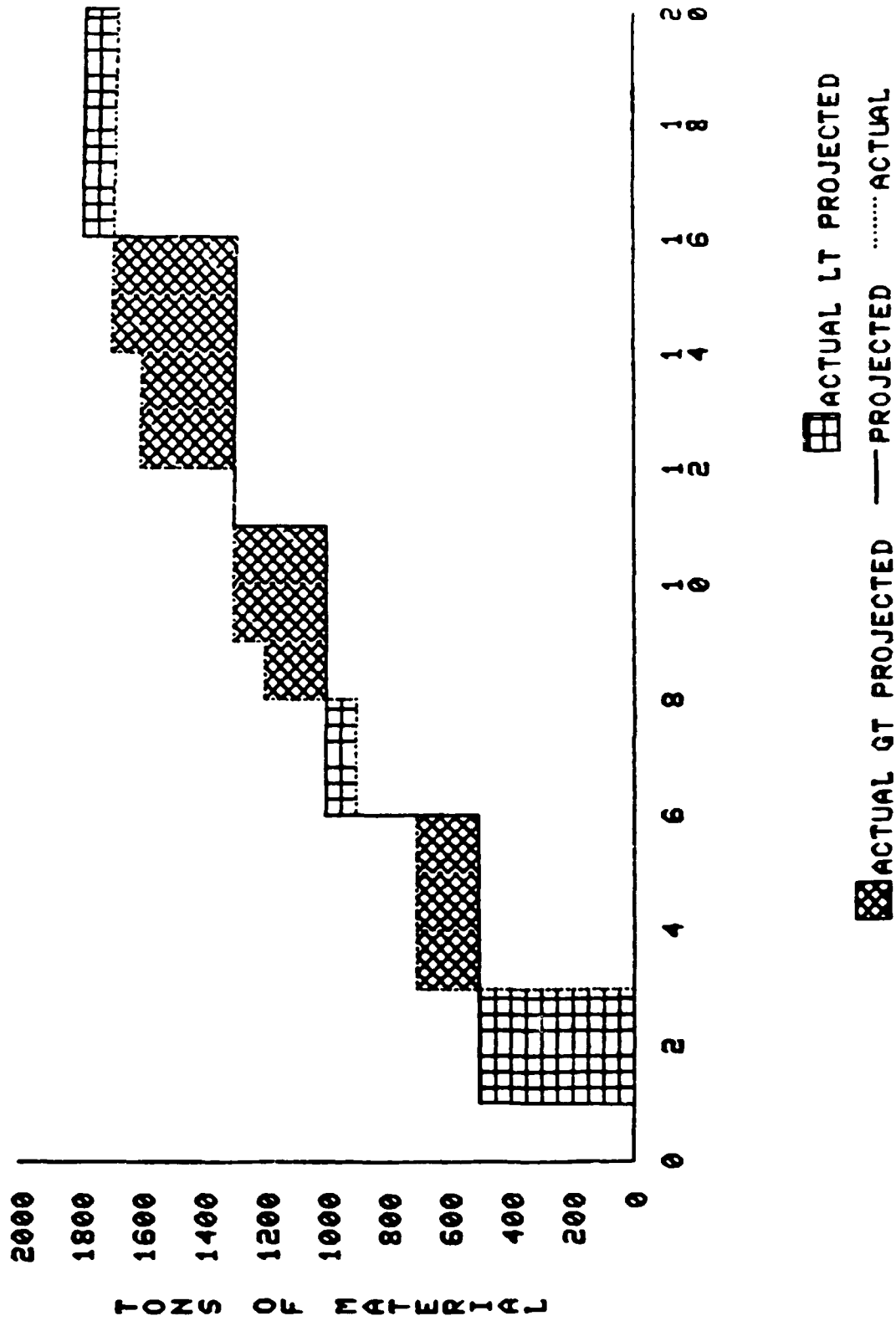


Figure 3-19. Step Graph With SHOW Option


```

USING    <vector mode>
PLANNED = <planned values> [<shade/fill options>]
[, ACTUAL = <actual values> [<shade/fill options>] ]
[, REQUIRED = <required value> [<shade/fill options>] ]
[; FOR <vector mode> [<vector list>] ]
[; PAGING <vector mode> [<vector list>] ]

```

The USING phrase defines the <vector mode> which identifies the set of numbers to be plotted. The <vector mode> is the usual ROWS, COLUMNS, SECTIONS or CATEGORY.

The <planned values> are the actual vector names used to identify the planned values. The syntax for <planned values> is as follows:

```
<start vector name> : <end vector name>
```

The <actual values> are similar to the <planned values> where the syntax would be as follows:

```
<start vector name> : <end vector name>
```

The <required value> is a single vector name used to identify a required value. Currently, this option is reflected on the display as a point-down triangle and may only utilize the COLOR option.

The FOR and PAGING phrases serve the same function as described for the bar graph in section 3.3.2.2.

The <shade/fill options> are:

```

SHADE    <shade option>
DENSITY  <density options>
FILL      <color name>
COLOR    <color option>

```

In order to utilize Gantt charts, the user must first create a tabular report which contains the data to be displayed and organized in the manner which the user desires it to be displayed. Refer to section 3.3.1 for more information on creating tabular reports. When issuing the DISPLAY GANTT statement, the user specifies which vector mode is to be keyed on for display purposes through the USING <vector mode> clause. Therefore, in the following Gantt parameter clause(s), the start and end values specified would be vector-header names that match the USING clause specification. For example, if a PCS statement said USING COLS, the vector-header names used to supply data for the gantt parameters must be column-header names. The following is a sample tabular report that could be utilized for Gantt charts:

	PB	PE	AB	AE	REQ
AIR	2	6	1	8	5
SEA	3	7	3	9	6
LAND	4	5	5	10	7

In this example, the rows have been used to denote AIR, SEA, and LAND forces and the columns have been used to designate the various time increments for force deployment. These time increments are determined by the user as they can be hours, days, weeks, or whatever is applicable to the specific display. Further, the header names are meaningful as they refer to the data contained therein: Planned Begin (PB), Planned End (PE), Actual Begin (AB), etc. As is shown here, both the Gantt PLANNED and ACTUAL clauses require two pieces of information from the matrix: a starting and ending point. The REQUIRED clause needs only a single vector for obvious reasons.

Now that the matrix has been generated, we are ready to display the data. The Gantt chart shown in figure 3-20 was created using the above matrix and the following PCS statement:

```
DISPLAY GANTT "TROOP MOVEMENT" (SIZE M,COLOR BLUE) X-TITLE "NUMBER OF WEEKS"
(SIZE M,COLOR RED) USING COLS PLANNED=PB:PE (SHADE UP,DEN LITE), ACT=AB:AE
(SHADE RIGHT,DEN HEAVY), REQ=REQ (COLOR GREEN); FOR ROWS.
```

3.3.2.8 Pie Charts. Pie charts are similar to bar graphs and line graphs in that they all derive their structure from the row and column values of the basic tabular report. The pie chart, however, does not contain the familiar x and y axis; therefore, the syntax structure is somewhat different:

```
DISPLAY PIE [CHART] [<pie chart label>] [<vector label options>] <chart
details>.
```

The word CHART may be added to improve readability. The <pie chart label> produces a textual message at the bottom of the pie chart. Its syntax is identical to the y-axis label of the bar graph:

```
OF "<literal>" [(SIZE <size option> ,COLOR <color option> )]
```

The <size option> establishes the size of the text. Valid options are SMALL, MEDIUM, LARGE and JUMBO. The default is the current size.

The <color option> establishes the color of the text. If no color is specified, the current color is used.

The <vector label options> controls the labeling of the pie wedges with the appropriate vector names. The syntax structure is:

UNCLASSIFIED EXERCISE SAMPLE 86

1359 02 MAY 86

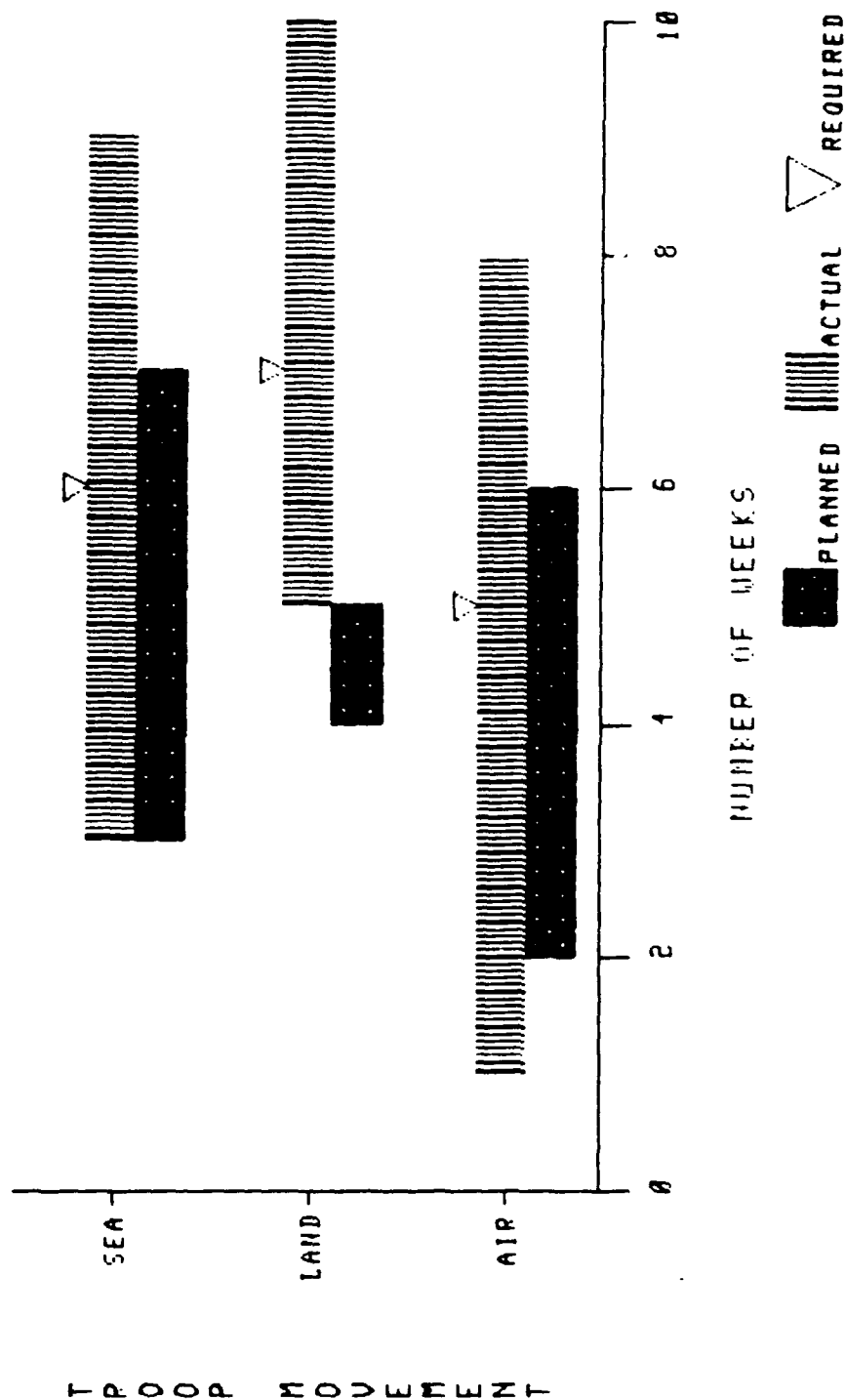


Figure 3-20. Basic Gantt Chart

```

[WITH] VALUE[S]      { FLOAT
                      JUSTIFY
                      SUPPRESS } OUTSIDE [LABELS] ,
{ SUPPRESS
  OBLIQUE
  HORIZONTAL }      INSIDE [LABELS] .

```

Observe on figure 3-21 that the pie chart displays the relative percentage value of each vector element. The percentage values are not found in the original tabular report, they were calculated automatically by GIPSY from the values extracted from the report. The WITH VALUE clause enables the user to display the actual matrix values along with the percent equivalent.

GIPSY's default is to display the vector names OUTSIDE of each individual wedge. The default is to FLOAT these outside labels (i.e., display the label in the proximity of the wedge). However, the user also has the option to JUSTIFY the labels so that they all line up with the right or left margins, or to SUPPRESS the outside labels so that they do not appear.

The placement of vector names INSIDE each wedge applies only to the USING vector. The inside labels are normally SUPPRESSED; however, the user can specify HORIZONTAL INSIDE LABELS which will write out the vector name horizontally, or OBLIQUE INSIDE LABELS which are written out perpendicular to a radius drawn through the middle of the wedge. If the label does not fit inside the wedge, it will be suppressed.

The syntax of <chart details> is similar to that for the other graphs:

```

USING <vector mode> [<vector name>] ["<legend name>"] [(<pie options>)]
[; FOR <vector mode> [<vector name list>]
[; PAGING <vector mode> [<vector name list>] .

```

The USING verb causes GIPSY to produce a separate wedge of the pie for each name in the USING vector mode. If the corresponding FOR vector mode contains more than one vector name, the USING and FOR vector labels will be displayed together, separated by a colon " : " (see figure 3-22).

The WEDGE option is a <vector name> detail which will cause the specified wedge to be set apart from the rest of the pie. More than one section of the pie can be wedged in a single report. The FILL option will cause the specified wedge to be filled with the specified color. The SUPPRESS option will suppress inside labels for the specified wedge.

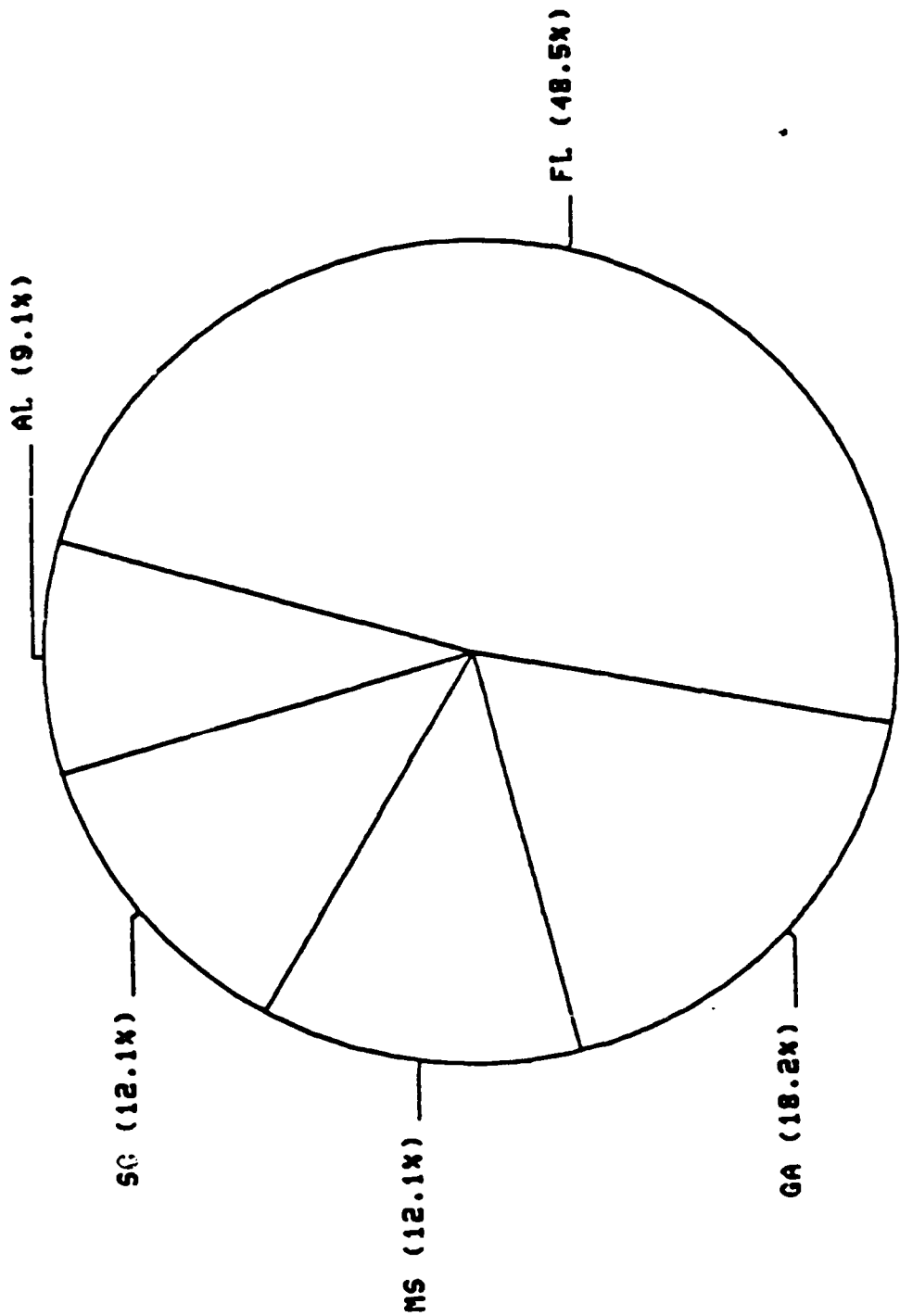
The <pie options> are:

```

{ WEDGE
  FILL <color name>
  SUPPRESS
}

```

UNCLASSIFIED
AIRFIELD TYPES BY STATE



MILITARY AIRFIELDS

Figure 3-21. Sample Pie Chart

UNCLASSIFIED
AIRFIELD TYPES BY STATE

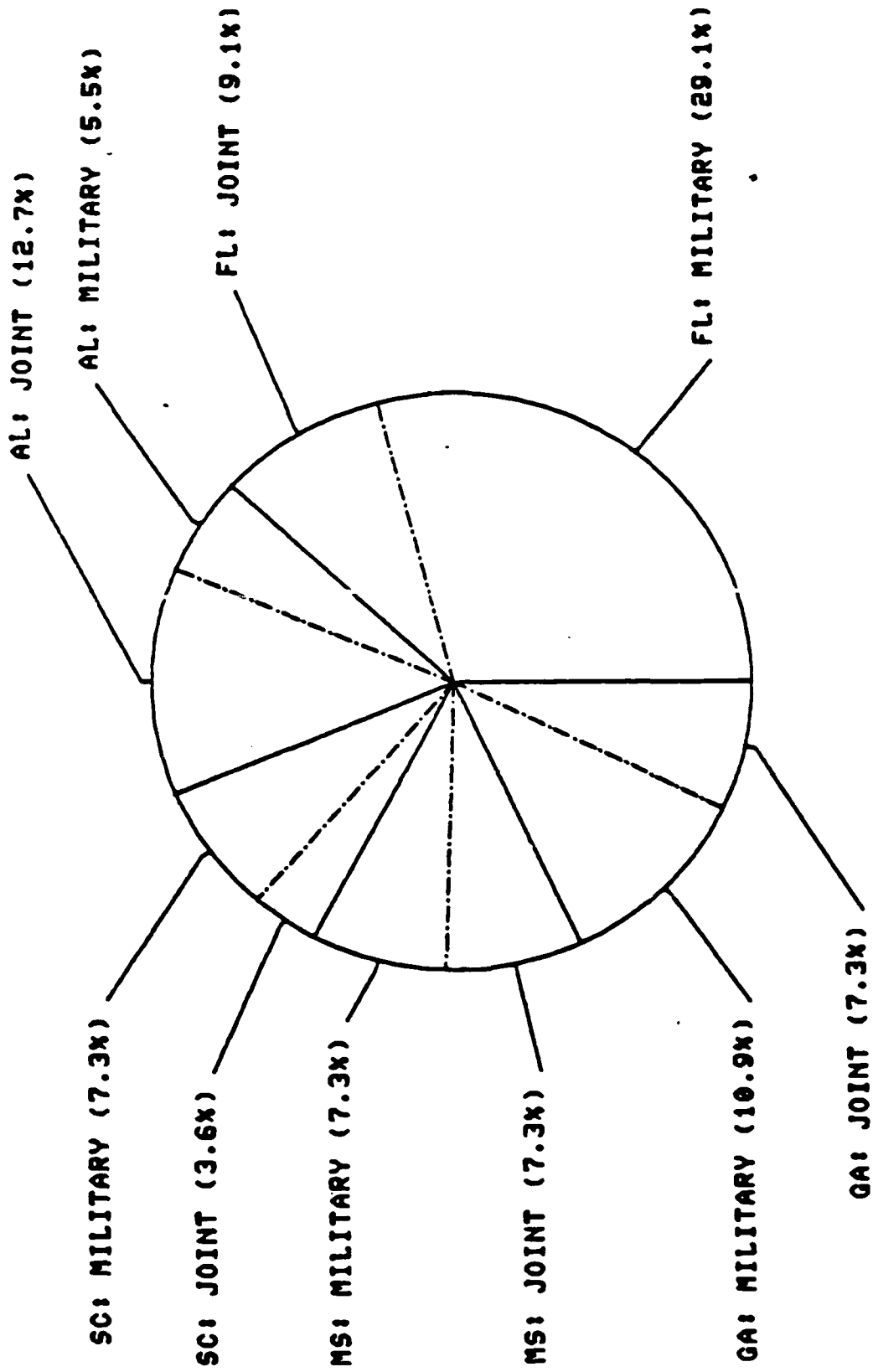


Figure 3-22. Pie Chart With USING and FOR Headers

The following tabular report was used to produce the pie charts in figures 3-21, 3-22, and 3-23:

	JOINT	MILITARY
AL	7	3
FL	5	16
GA	4	6
MS	4	4
SC	2	4

Figure 3-21 was produced by the command:

DISPLAY PIE OF "MILITARY AIRFIELDS" USING ROWS; FOR COLUMN MILITARY.

Note that the FOR header label is suppressed if only one vector is used.

Figure 3-22 was produced using:

D PIE USING ROWS.

Note that a separate wedge is formed for each ROW header, which is then divided up between each COLUMN header.

Figure 3-23 was produced by the statement:

DISPLAY PIE WITH VALUE, JUSTIFY OUTSIDE LABELS,
HORIZONTAL INSIDE, USING ROWS AL,FL(WEDGE),GA,MS,SC.

Note that the FL section is wedged out. Also observe that the integer values which appear beneath each label is the corresponding element value from the original tabular report.

3.3.3 Modifying Statistical Reports. GIPSY provides the user the means of modifying the basic reports discussed in section 3.3.2. This includes modifying the matrix values, changing default graph parameters, and adding symbols and text to the graphic report.

3.3.3.1 Matrix Modification. GIPSY provides the user with several basic commands to modify the values contained in the matrix. This includes the ability to produce vector totals, accumulate vector values, and to restrict the number of decimal places displayed. For more extensive modifications of the matrix; see section 3.3.7, Building a New Report.

3.3.3.1.1 Limiting Rows, Columns, Sections and Categories. The defaults in GIPSY are designed to always use all of the users data that may be addressed by a user specified statement. Occasionally, it is desirable to limit those defaults to a smaller subset of the users data. GIPSY provides a mechanism called LIMIT to perform that function.

UNCLASSIFIED
AIRFIELD TYPES BY STATE

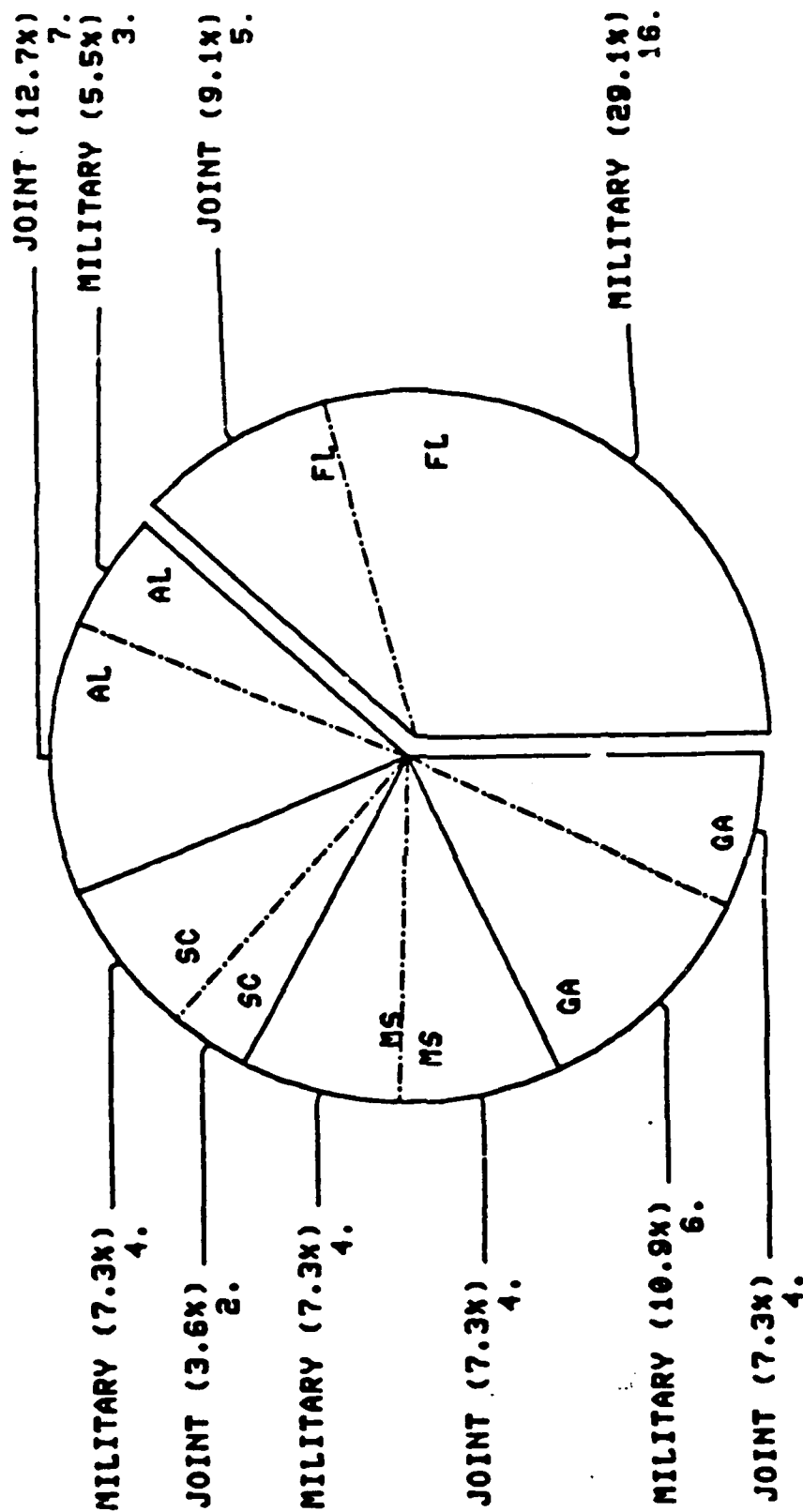


Figure 3-23. Pie Chart With LABEL Options

The syntax for the LIMIT statement is:

```
LIMIT { ROWS
        COLUMNS
        CATEGORIES
        SECTIONS } { <list of vector names>
                     TO ALL
                     * } [;< element condition>].
```

The <list of vector names> referenced above may be a single vector name or a list of names separated by commas. The list may include ranges in the form:

<first vector desired> THRU <last vector desired>.

ROWS, COLUMNS, CATEGORIES, and SECTIONS identify the vector mode to which the limit is to apply. TO is an optional noise word added for clarity of the limit function. The <first vector desired> and <last vector desired> are defined as the name implies. The THRU operator causes all vectors in between (inclusive) to be included in the limit. If the last preceded the first they will be applied in reverse order.

The LIMIT statement limits all system defaults which use the referenced vector mode to only the named vectors. Another effect of the limit is the effective reordering of the vectors to the order in which they appear in the LIMIT statement. The specified limit will remain in effect for all applicable graph and report statements until it is either changed or a new report is accessed, or the TRANSFER statement is issued.

A few examples will clarify this. Given this report:

	CA	CB	CC	CD	CZ	CQ	ZZ	AA
R1	1	2	3	4	5	6	7	8
R2	9	10	11	12	13	14	15	16
R3	17	18	19	20	21	22	23	24
R4	25	26	27	28	29	30	31	32
R5	33	34	35	36	37	38	39	40
R6	41	42	43	44	45	46	47	48

The following statements:

```
LIMIT ROWS TO R1, R4, R6.
LIMIT COLS TO CC, ZZ THRU CD.
DISPLAY REPORT.
```

will result in the report:

	CC	ZZ	CQ	CZ	CD
R1	3	7	6	5	4
R4	27	31	30	29	28
R6	43	47	46	45	44

Note the effect of the order in which the vectors appear in the report. Since the THRU pair was specified in reverse order, the vectors show up in the report in reversed order. Rows R2, R3, and R5 and columns CA, CB, and AA will not be picked up in any GIPSY display commands unless specifically referenced. If the vectors were LIMITED or specifically referenced, they will be used; otherwise, they don't exist for GIPSY. All vectors are always available and may be re-included by resetting the limit to all vectors. For example:

```
LIMIT ROWS TO ALL.
LIMIT COLUMNS *.
DISPLAY REPORT.
```

will produce the original report with which we started. The asterisk is a synonym for "TO ALL".

The <element condition> is a set of conditions which must be satisfied to have the associated limit function performed. Its specific syntax is:

```
IF { ANY } ELEMENTS <relational operator> <value>
   { ALL }
   { AND }
   { OR } ELEMENTS <relational operator> <value>
```

The statement:

```
LIMIT ROWS TO ALL IF ANY ELEMENT GT 20.
```

will cause every row to be included in the report as long as there is at least one element in that row with a value greater than 20.

The limit functions may be applied to categories and sections in the same manner as we have illustrated for rows and columns. For example, the following report could show officer turnover by grade within service for agencies by fiscal year:

		ARMY				NAVY				USAF			
		0-1	0-2	0-3	0-4	0-1	0-2	0-3	0-4	0-1	0-2	0-3	0-4
FY80													
	OJCS	12	10	3	2	10	8	1	2	10	8	6	4
	CCTC	14	11	6	3	12	11	7	3	14	12	10	8
	DIA	10	9	3	1	10	8	4	6	11	7	5	3
	WHCC	8	8	8	8	3	4	5	6	2	4	6	8
FY81													
	OJCS	13	12	8	6	14	8	4	2	15	13	12	10
	CCTC	15	15	12	9	15	7	3	3	14	13	11	12
	DIA	12	10	10	9	10	8	4	4	11	9	7	5
	WHCC	14	14	14	14	12	6	5	4	10	10	0	8

We can reduce the report to the FY80 section and NAVY category by issuing the statements:

LIMIT SECTION TO FY80. LIMIT CATEGORY TO NAVY. DR.

The resulting tabular report would be:

FY80	NAVY			
	0-1	0-2	0-3	0-4
OJCS	10	8	1	2
CCTC	12	11	7	3
DIA	10	8	4	6
WHCC	3	4	5	6

The interrupt command //LIMIT <vector mode> can be used to list the names of the vectors currently in the limit.

3.3.3.1.2 Report Totals. Report totals for any of the four vector modes can be added or removed from the tabular report once the tabular report is built. The APPEND command is provided to append a total vector as a new vector in the report. The REMOVE statement is used to delete the totals from the tabular report.

{ APPEND REMOVE }	ROW	↑ 0	TOTALS [NAMED <name for total>]
	COLUMN		
	CATEGORY		
	SECTION		

The command APPEND ROW TOTALS will cause GIPSY to add together all the values in each row to produce a new column of totals. The new column will be given the name specified on the APPEND command. If the NAMES parameter is not provided, the name will default to TOTAL. Similarly, column totals will be appended as a new row; category totals will be added as a new section; and, section totals will be added as a new category when the appropriate command is issued. If a vector already exists by the name to be added, GIPSY will notify you that the vector already exists and that vector mode total will not be produced. Once the total is added to the report, GIPSY does not distinguish it from any other vector.

If a vector mode is not specified, all modes currently defined in your tabular report will be totaled. Note that if a LIMIT command (section 3.3.3.2) is in effect for any vector mode the totalling action will only be performed for those vectors specified in the LIMIT. Figure 3-24 contains several examples appending various totals.

The REMOVE command causes an appended or preexisting total vector to be deleted. Its structure is identical to the APPEND except that it removes the total vector rather than adding it. If you named the vector in appending it, you must also provide that name in removing it.

DISPLAY REPORT.

	CL1	CL2	CL3
RU1	1	2	3
RU2	4	5	6

1 OF 1

APPEND ROW TOTALS. DISPLAY REPORT.

	CL1	CL2	CL3	TOTAL
RU1	1	2	3	6
RU2	4	5	6	15

1 OF 1

For the cases with SECTIONS and CATEGORIES: DISPLAY REPORT.

		CT1			CT2		
		CL1	CL2	CL3	CL1	CL2	CL3
SC1							
RU1		1	2	3	100	200	300
RU2		4	5	6	400	500	600
SC2							
RU1		10	20	30	1000	2000	3000
RU2		40	50	60	4000	5000	6000

1 OF 1

APPEND TOTALS. DISPLAY REPORT.

		CT1				CT2				TOTAL			
		CL1	CL2	CL3	TOTAL	CL1	CL2	CL3	TOTAL	CL1	CL2	CL3	TOTAL
SC1													
RU1		1	2	3	6	100	200	300	600	101	202	303	606
RU2		4	5	6	15	400	500	600	1500	404	505	606	1515
TOTAL		5	7	9	21	500	700	900	2100	505	707	909	2121
SC2													
RU1		10	20	30	60	1000	2000	3000	6000	1010	2020	3030	6060
RU2		40	50	60	150	4000	5000	6000	15000	4040	5050	6060	15150
TOTAL		50	70	90	210	5000	7000	9000	21000	5050	7070	9090	21210
TOTAL													
RU1		11	22	33	66	1100	2200	3300	6600	1111	2222	3333	6666
RU2		44	55	66	165	4400	5500	6600	16500	4444	5555	6666	16665
TOTAL		55	77	99	231	5500	7700	9900	23100	5555	7777	9999	23331

1 OF 1

REMOVE SECTION TOTALS. DR.

		CT1				CT2			
		CL1	CL2	CL3	TOTAL	CL1	CL2	CL3	TOTAL
SC1									
RU1		1	2	3	6	100	200	300	600
RU2		4	5	6	15	400	500	600	1500
TOTAL		5	7	9	21	500	700	900	2100
SC2									
RU1		10	20	30	60	1000	2000	3000	6000
RU2		40	50	60	150	4000	5000	6000	15000
TOTAL		50	70	90	210	5000	7000	9000	21000
TOTAL									
RU1		11	22	33	66	1100	2200	3300	6600
RU2		44	55	66	165	4400	5500	6600	16500
TOTAL		55	77	99	231	5500	7700	9900	23100

1 OF 1

Figure 3-24. Appended Totals

3.3.3.1.3 Vector Sequencing. It is often useful to be able to number your rows, columns, sections and categories for identification purposes or for curve, step or gantt graph functions.

GIPSY provides a mechanism for sequencing the vectors using the following syntax:

$\left\{ \begin{array}{c} \text{APPEND} \\ \text{REMOVE} \end{array} \right\}$	$\left[\begin{array}{c} \text{ROW} \\ \text{COLUMN} \\ \text{CATEGORY} \\ \text{SECTION} \end{array} \right]$	SEQUENCES [NAMED <new sequence name>]
--	--	---------------------------------------

Note that this function uses the APPEND/REMOVE command described for handling report totals. All syntax and semantics is the same as for the totals except that it produces/removes a sequence vector.

3.3.3.1.4 Accumulating Vector Values. GIPSY provides the capability of converting a sequence of vector values into a running total, where each vector element equals the sum of the previous elements.

The syntax for this capability is:

$\left\{ \begin{array}{c} \text{ACCUMULATE} \\ \text{DIFFERENCE} \end{array} \right\}$	$\left\{ \begin{array}{c} \text{ROW} \\ \text{COLUMN} \\ \text{CATEGORY} \\ \text{SECTION} \end{array} \right\}$	[<vector list>]
;FOR	$\left\{ \begin{array}{c} \text{ROW} \\ \text{COLUMN} \\ \text{CATEGORY} \\ \text{SECTION} \end{array} \right\}$	[<vector list>] [;FOR...]

The first <vector list> specifies which vector headers will contain the accumulated values. The FOR option is used to limit the elements which are involved in the calculations.

The following matrix shows an earnings report for a four month period:

	INCOME	EXPENSES	PROFIT
JAN	3000	2500	500
FEB	2300	3200	-900
MAR	2400	2350	50
APR	5000	3230	1770

By entering the command "ACCUMULATE COLUMNS." the following matrix is produced. Note that each row is an accumulation of previous rows.

	INCOME	EXPENSES	PROFIT
JAN	3000	2500	500
FEB	5300	5700	-400
MAR	7700	8050	-350
APR	12700	11280	1420

The command DIFFERENCE has the reverse effect of ACCUMULATE.

3.3.3.1.5 Limiting Number of Decimal Places. GIPSY provides the user with the capability to limit the number of decimal places to be displayed for tabular reports, bar graphs, line and point graphs, pie charts, and histograms by using the SET command.

The syntax of the SET command is:

$$\text{SET} \quad \left\{ \begin{array}{c} \text{MAXDEC} \\ \text{FIXDEC} \end{array} \right\} \quad \left\{ \begin{array}{c} \langle n \rangle \\ * \\ \text{ALL} \end{array} \right\} .$$

When displaying values on graphs, GIPSY will normally display all of the decimal places associated with that value. In the case of tabular reports, each value in a column will contain the same number of decimal places as that of the longest decimal value.

The optional parameters MAXDEC and FIXDEC allows the standard GIPSY format to be overridden. MAXDEC <n> declares that no value in the display shall have more than n digits to the right of the decimal point. GIPSY will continue to decide for those values which will have less than the specified value of n. The parameter FIXDEC declares that every entry in the display shall have the fixed number of decimal places as specified. The command:

SET FIXDEC 0

will cause the entire display to consist of integers regardless of the fractional parts. Every number will be rounded to the nearest whole number.

The MAXDEC or FIXDEC options remain in effect for the entire GIPSY session unless they are reset to the normal configuration using the ALL or asterisk "*" options.

3.3.3.2 Establishing Default Graph Parameters. The DEFAULT command allows the user to establish default parameters that will be used in place of the system supplied parameters, when displaying a graph. The command allows the user to specify colors to be used for fill options or specific shading patterns for a bar graph, histogram, or Gantt chart or to designate the x and y axis proportion for a curve or step graph. The syntax for the command is:

$$\text{DEFAULT} \quad \left\{ \begin{array}{l} \text{FILL} \langle \text{fill options} \rangle \\ \text{SHADING} \langle \text{shade options} \rangle \\ \text{X:Y} \text{ proportion} \end{array} \right\} .$$

Each statement is in effect for only the following graph unless a global or specific LOCK statement is issued (see section 3.3.3.3). An UNLOCK statement is then required to terminate its use. If the DEFAULT FILL or DEFAULT SHADING is locked, it may be temporarily overridden by specifying the color fill or shading of each vector individually in a DISPLAY statement.

The various DEFAULT options and the LOCK command are discussed in the following sections.

3.3.3.2.1 Fill Specifications. In the DEFAULT FILL command, the color names and the sequence in which the fill colors will be used in displaying a bar graph, histogram, step graph or pie chart are identified. The syntax is:

DEFAULT FILL <color fill list>

where <color fill list> has the format <color 1> [, <color 2>, ...].

If there are more vectors to be filled than there are color names specified in the fill list, the list will be recycled.

The default for GIPSY is to have bar or step graphs shaded instead of color filled.

The SET <graph type> FILL ON command (see section 3.3.3.2.5) is required to activate the use of the DEFAULT FILL colors.

3.3.3.2.2 Shading Specifications. In the DEFAULT SHADING command the combinations of shade pattern, density and color to be used to distinguish each vector in a bar graph, step graph, or histogram are defined. The syntax is:

DEFAULT SHADING <shade options>

where <shade options> are ($\left[\begin{array}{l} \text{SHADE } \langle \text{options} \rangle \\ \text{DENSITY } \langle \text{options} \rangle \\ \text{COLOR } \langle \text{color name} \rangle \end{array} \right]$) (<options>), ...]

The SHADE options are the same as those available for in-line specification: LEFT, RIGHT, ACROSS, UP, DOWN, NONE. More than one shading direction may be specified by separating each option with a comma or the word AND. The DENSITY <option> includes: LIGHT, MEDIUM, HEAVY. The COLOR <color name> defines the color in which the bar outline and shading will be drawn. A single shading definition is enclosed in parentheses. The number of shading definitions, each separated by a comma, is determined by the user. If the number of vectors to be shaded exceeds the shading definitions, the list of definitions is recycled.

Use of the DEFAULT SHADING command in conjunction with the DEFAULT FILL command can be controlled by the SET <graph type> SHADE command (see section 3.3.3.2.5).

3.3.3.2.3 Proportion Specifications. The DEFAULT X:Y statement is used only for the curve and step graphs. It allows the user to specify the proportion between units on the x-axis and units on the y-axis. The syntax is:

DEFAULT X:Y = <m:n>

This determines that <m> units on the x-axis will be the same length as <n> units on the y-axis. Both <m> and <n> must be specified; they may be real or integer constants.

The old proportion command, which produces the same results as the DEFAULT command, can still be used. The syntax is:

X:Y = <m:n>

3.3.3.3 Preserving User Supplied Defaults. GIPSY provides LOCK/UNLOCK capabilities for preserving options of the DEFAULT command, such as FILL, SHADING, and PROPORTION. The LOCK statement will preserve the last specified FILL, SHADING, or PROPORTION values and activate them for all subsequent displays until they are unlocked. In-line commands for the FILL and SHADING options will, however, temporarily override any LOCK command in effect. Subsequent displays will have all options locked that were locked prior to the in-line command. When unlocked, GIPSY defaults will apply unless overridden by temporary graph options or a new LOCK command. The syntax for this capability is:

$\left\{ \begin{array}{l} \text{LOCK} \\ \text{UNLOCK} \end{array} \right\}$	$\left[\begin{array}{l} \text{FILL} \\ \text{SHADING} \\ \text{SHADE} \\ \text{PROPORTION} \end{array} \right]$
--	--

All parameters will be locked or unlocked if no options are specified.

Locking more than one parameter requires separate statements such as

LOCK FILL.
LOCK SHADING.

3.3.3.3.1 Control of Fill vs. Shading. When producing a bar or step graph or histogram, the default is for GIPSY to differentiate between the bars by using different shading patterns. Obtaining color filled bars requires user override of this default. This can be accomplished by in-line FILL options in the DISPLAY <graph> command, or by using the command:


```
SET AUTO { BAR STEP } { SHADE FILL } { ON OFF } .
```

The default is SHADE ON and FILL OFF. If SET BAR FILL ON is issued, all subsequent bars will be color filled rather than shaded. The colors will be obtained from the default color table or from the DEFAULT FILL command, if supplied. Entering FILL ON followed by SHADE ON will cause the bars to be both color filled and shaded. Pie charts cannot be shaded; however, automatic color fill can be controlled by:

```
SET [AUTO] PIE FILL { ON OFF } .
```

3.3.3.3.2 Control of Axis Parameters. The AXIS command provides user control over various components of the axis.

The syntax of this command is:

```
AXIS [ X ] { SCALE <scale options>
      Y   { TIC   <tic options>
          GRID  <grid options> }
```

Both the x- and y-axis may be individually defined for the SCALE, TIC, and GRID options. The y-axis definition is valid for any graph; the x-axis scale definition only applies to the CURVE and STEP graphs as well as GANTT charts, and the x-axis TIC and GRID does not apply to bar graphs. If neither the x- or y-axis is indicated the specified options will be set for both axis.

3.3.3.3.3 Establishing Axis Parameters. GIPSY automatically scales all graphs to fit the range of the data values being plotted. There are, however, occasions when it is desirable to manually control the limits and increments of the axis scaling. The syntax of this command is:

```
AXIS [ X ] SCALE { FROM <lower limit>
                  TO   <upper limit>
                  BY   <increment> }
```

The y-axis may be scaled for any graph; the x-axis scaling affects the CURVE and STEP graphs as well as GANTT charts. The effect of the AXIS SCALE command lasts until the next graph is generated. The specified AXIS SCALE definition may be preserved to be applied to all appropriate graphs subsequently generated by issuing a LOCK statement which is discussed in section 3.3.3.3.6 GIPSY will automatically supply defaults for any omitted parameter.

GIPSY automatically scales the y-axis based on the range from the minimum data value to the maximum data value. Consequently, graphs do not necessarily start from an origin of zero. The AXIS SCALE statement may be used to force the origin to be zero (or any other value). For example:

AXIS Y SCALE FROM 0.

will cause the next graph to be generated with y-axis origin of zero. If the specified y-axis scale (from minimum to specified maximum) is not evenly divisible by the increment, the graph will be drawn up to the specified upper limit, but no label will be present for the top of the y-axis.

Figure 3-25 was produced by the commands:

```
AXIS Y SCALE FROM 0 TO 25000 BY 2500.  
DISPLAY BAR WITH VALUES USING COLS.
```

The old SCALE command, which produces the same results as the AXIS SCALE command, can still be used. The syntax is:

```
SCALE  $\begin{bmatrix} X \\ Y \end{bmatrix}$  AXIS [FROM <lower limit>] [TO <upper limit>] [BY <increment>].
```

The AXIS SCALE statement is often used in conjunction with the LOCK statement to produce a set of graphs in which data has differing ranges, but it is desirable to force all graphs in the set to have the same scaling.

3.3.3.3.4 Adding Tic Marks. The AXIS TIC command will cause the graph axis to be displayed with user specified tic marks. The syntax for this command is:

```
AXIS  $\begin{bmatrix} X \\ Y \end{bmatrix}$  TIC <n>  $\left\{ \begin{array}{l} \text{OUTSIDE} \\ \text{INSIDE} \\ \text{COLOR <color>} \end{array} \right\}$ .
```

The user is able to individually control the tic marks for each axis. Each axis increment is divided into <n> equal parts. If an increment of 0 is supplied the system default of no tic marks will be reinstated. Subsequent displays can retain the same TIC capability by issuing the LOCK TIC command, discussed in section 3.3.3.3.6. The default OUTSIDE parameter causes the TIC marks to be drawn on the label side of the axis. The INSIDE parameter causes the tic marks to be drawn on the data side of the graph. Both may be specified concurrently. The COLOR parameter can be used to specify the color of the TIC marks.

Figure 3-26 was produced by the commands:

```
AXIS Y TIC 5 OUTSIDE, COLOR RED.  
AXIS X TIC 10 INSIDE, COLOR BLUE.
```

3.3.3.3.5 Adding Grid Lines. The AXIS GRID command will cause the specified grid lines to be drawn perpendicular to the specified axis. The syntax for this command is:

AXIS Y SCALED FROM 0 TO 25000 BY 2500

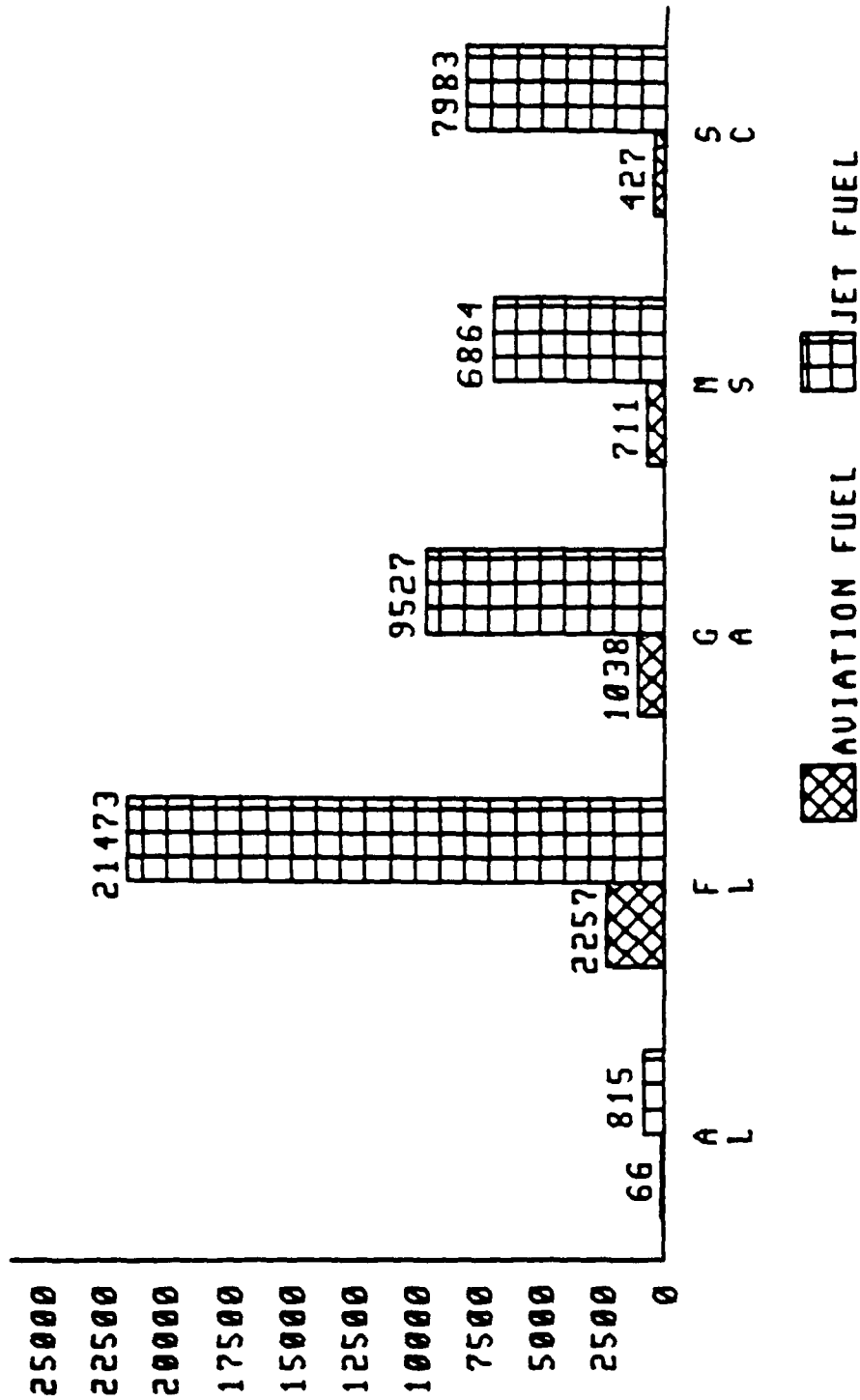


Figure 3-25. Result of SCALE Command

TIC MARK CAPABILITY FOR X AND Y AXES

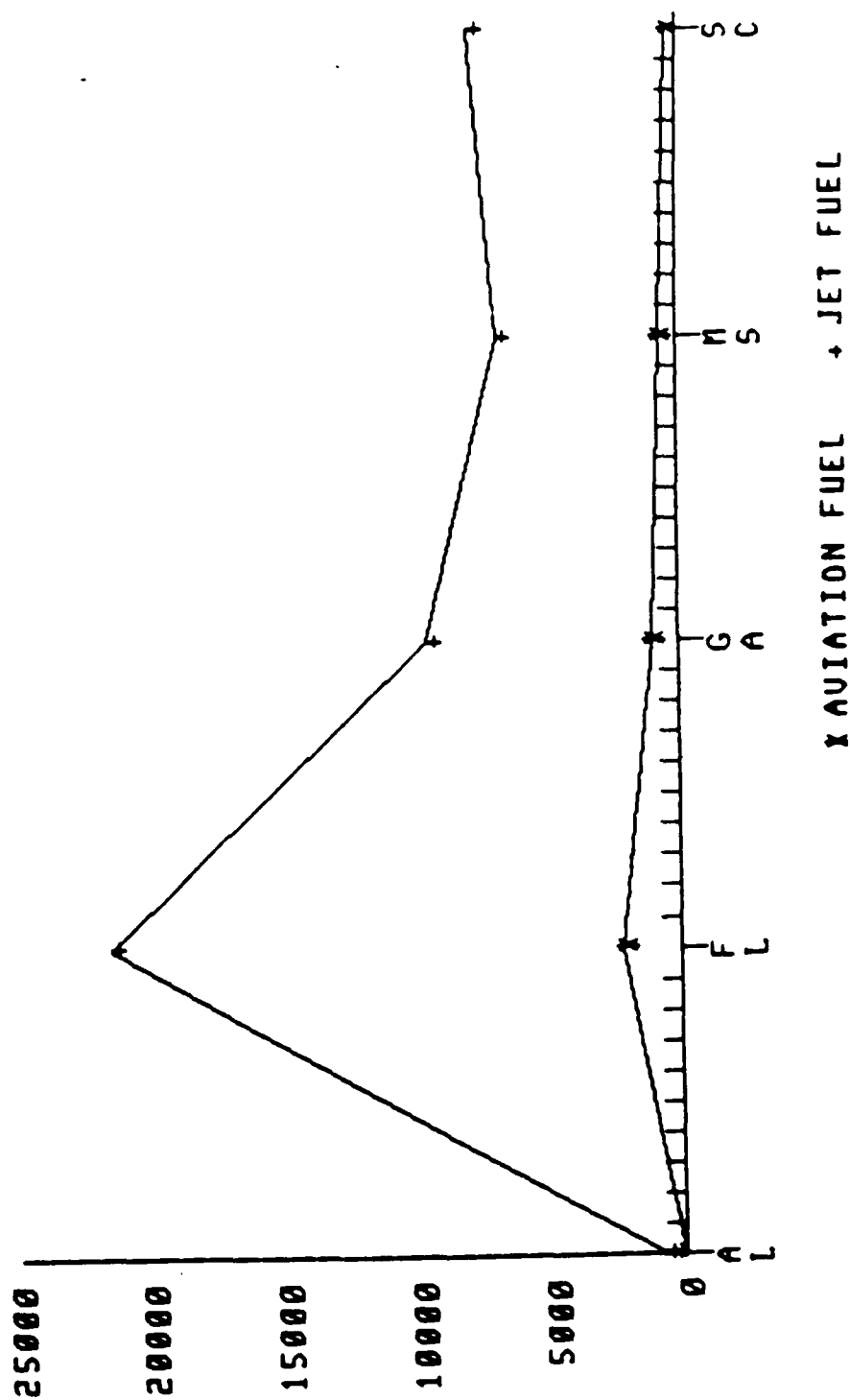


Figure 3-26. Result of TIC Command

AXIS $\begin{bmatrix} X \\ Y \end{bmatrix}$ GRID $\left\{ \begin{array}{c} <n> \\ / \\ <n> \end{array} \right\}$ $\begin{bmatrix} \text{LINE } <\text{line type}> \\ \text{WIDTH } <\text{width}> \\ \text{COLOR } <\text{color}> \end{bmatrix}$

Each interval defined by axis labels will be subdivided into $<n>$ increments. If the command:

AXIS Y GRID 3.

is issued, the y-axis will have 2 grid lines between each label. If the increment is preceded with a slash "/" the grid occurs every $<n>$ labels. The command:

AXIS Y GRID /3.

will cause the y-axis to have a grid line once every three labels.

The LINE, WIDTH, and COLOR options give the user the opportunity to choose the line type, line width and color of the grid lines. Subsequent displays can retain the same GRID capability by issuing the LOCK GRID command, discussed in section 3.3.3.3.4.

Figure 3-27 was produced by the commands:

AXIS Y GRID 3 LINE DASHED, COLOR RED.
 AXIS X GRID /2 LINE DOTTED, WIDTH 3 COLOR YELLOW.

3.3.3.3.6 Preserving Axis Parameters. GIPSY provides a LOCK/UNLOCK statement to preserve axis parameters such as SCALE, TIC and GRID. The lock statement will cause the last specified parameter(s) to be activated for all subsequent displays, until they are unlocked.

When unlocked, GIPSY defaults will apply unless overridden by temporary graph options or a new LOCK command. LOCKING or UNLOCKING X- or Y-AXIS (i.e., LOCK X-AXIS. or LOCK Y-AXIS.) will cause all X or Y AXIS options to be locked or unlocked.

The syntax for this capability is:

$\left\{ \begin{array}{c} \text{LOCK} \\ \text{UNLOCK} \end{array} \right\}$ $\begin{bmatrix} X \text{ [AXIS]} \\ X\text{-AXIS} \\ Y \text{ [AXIS]} \\ Y\text{-AXIS} \end{bmatrix}$ $\begin{bmatrix} \text{SCALE} \\ \text{TIC} \\ \text{GRID} \end{bmatrix}$

If LOCK or UNLOCK is specified with no options (i.e., LOCK. or UNLOCK.) all user specified graph options will either be locked or unlocked.

Locking more than one parameter requires separate GIPSY statements such as:

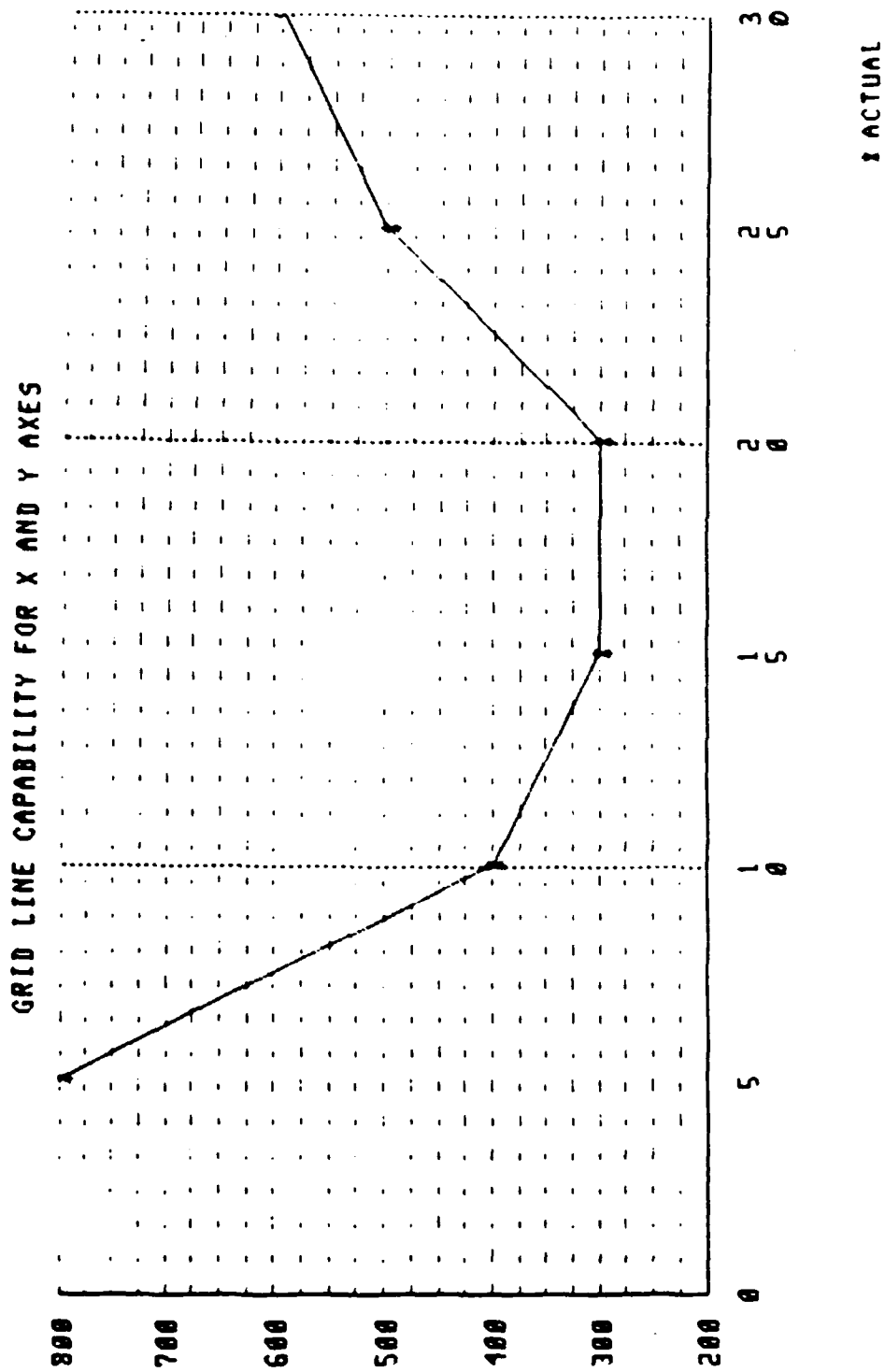


Figure 3-27. Result of GRID Command

LOCK X SCALE.
LOCK X TIC.

3.3.4 Enhancing Statistical Reports. Enhancing reports refers to the ability to modify or add information to the graph that is not matrix oriented. This includes the addition of symbols and textual data which can be placed anywhere on the graph, re-displaying a report or displaying the report without the graph.

3.3.4.1 Adding Symbol. Symbols may be added to the graphic display in order to enhance the final report. A symbol is any alphanumeric string up to 132 characters in length. The string may also contain GIPSY special graphic characters (see section 3.4.2.1). The syntax for this command is:

$$\left\{ \begin{array}{l} \text{DISPLAY} \\ \text{D} \\ \text{GENERATE} \\ \text{G} \end{array} \right\} \left\{ \begin{array}{l} \text{SYMBOL} \\ \text{S} \end{array} \right\} \left[\begin{array}{l} "<\text{symbol name}>" \\ "<\text{symbol}>" [<\text{symbol options}>] \end{array} \right]$$

A typical use of this command would be

DISPLAY SYMBOL "ABC".

Execution of this command will activate the graphic cursor, which the user would then move to the area on the screen where he wishes to display the symbol. Depressing the space bar will cause the symbol to be displayed, placing the lower left corner of the first character at the cursor location. Depressing an asterisk causes the cursor to remain activated so that the symbol may be displayed at multiple locations.

The DISPLAY option allows the symbol to be displayed for the current graph only. The symbols will reappear if the graph is refreshed (see section 3.3.4.3).

The GENERATE option will cause the symbol to be displayed on all subsequent graphs, even if the graph type is changed. The use of the generated symbols can be controlled, however, through the use of the LIMIT command:

$$\text{LIMIT SYMBOLS TO} \left\{ \begin{array}{l} \text{NONE} \\ \text{ALL} \\ * \\ <\text{list of defined symbols}> \end{array} \right\}$$

where the asterisk "*" is equivalent to ALL.

The <symbol options> provide additional capabilities to be used when displaying the symbols. They are:

NAME <symbol name>	Attaches a name to the symbol so that it can be referenced by the LIMIT command.
COLOR <color>	Allows the user to specify the color of the symbol.
SIZE <size option>	Allows the user to specify the character size of the symbol.
CENTER	Positions the symbol so that the first character is centered over the cursor.

If multiple options are used, they must be separated by a comma.

3.3.4.2 Adding Textual Information. Once a GIPSY display has been produced, it is possible to add blocks/groups of textual information to the graphic report. This textual information may be predefined when given a name and later displayed by referencing the name assigned to it. The syntax for this block structure is as follows:

```
BUILD TEXT TABLE [(CONTINUED)].
TEXT NAME[D] <name> [(<group options>)]
"<text>" [(<line options>)] [; "<text>"...].
TEXT NAME[D] <name> ...
.
.
.
END.
```

The command BUILD TEXT TABLE establishes the build mode for the text table. Using the CONTINUE option will cause new text structures to be added to the existing text table. Individual blocks/groups of text must be named. There is no limit on the number of textual groups defined in the table. The command TEXT NAME[D] establishes the start of text that will be grouped together at display time. The text <name> must be unique within the TEXT TABLE and may be up to 12 alphanumeric characters long.

The <group options> are listed below. Individual options are separated with commas.

BOX[ED]	is the system default. All textual information will have a box drawn around it regardless of the number of lines associated with it.
NO[T] BOX[ED]	will suppress the textual box from being drawn.
LINE <line option>	will determine the type of line the textual box will be drawn in. Line options are <u>SOLID</u> , DOTTED, LDASHED, DASHED and DOT-DASHED with <u>SOLID</u> being the default.

WIDTH <n> will determine the width of the line the textual box will be drawn in. Values are between 1 and 15.

COLOR <color> identifies the color of the textual box lines and all text associated with the box. If no color is specified the current color is used.

SIZE <size option> establishes the size of the textual output. Valid options are SMALL, MEDIUM, LARGE, and JUMBO. The default is the current size.

BLANK

ED
ING

 will cause the area within the text box to be color filled with the background color.

Any <group option> specified after the declarative TEXT NAME[D] <name> command establishes the default for all lines of text associated with its name. Each line of textual input is entered as follows:

"<text>" [(COLOR <color>, SIZE <size option>)] [; "<text>"].

There is no limit, other than terminal constraints, to the number of lines that can be grouped under one <name>. The use of a semicolon signals continuation of textual input and the period serves as a terminator for the current text name. Every line of text can have its own COLOR or SIZE options, which will override the default values.

The command END terminates the build mode and completes the TEXT TABLE block structure.

The following is a sample Text Table. Notice that two groups/blocks of text have been declared: TXT1 and TXT2.

BUILD TEXT TABLE.

TEXT NAMED TXT1 (NOT BOXED, COLOR RED, SIZE SMALL)

"THIS IS TEXT1";
 "DEMONSTRATING THE FOLLOWING:";
 "THE NOT BOXED OPTION";
 "COLOR RED";
 "SIZE SMALL FOR ALL TEXT".

TEXT NAMED TXT2 (COLOR BLUE)

"THIS IS TXT2" (SIZE SMALL);
 "DEMONSTRATING THE FOLLOWING" (SIZE LARGE);
 "DEFAULT FOR BOXED AND BOX LINE TYPE" (SIZE SMALL);
 "INDIVIDUAL SIZE OPTION FOR EACH LINE" (SIZE JUMBO);
 "COLOR BLUE FOR ALL" (SIZE SMALL).
 END.

These textual items can be accessed and displayed after any graphic output.

The command:

DISPLAY TEXT NAME[D] <name> .

will activate the graphic cursor causing graphic cross-hairs to appear on the screen. Once you have positioned the graphic cursor to the position desired, depress the space bar. The textual information named in the DISPLAY TEXT command will be displayed positioning the first character of the first line of text at the position indicated by the cross hairs. Entering an asterisk "*" instead of the space bar will cause the graphic cursor to remain activated, allowing the user to display the text at multiple locations.

Figures 3-28 through 3-32 demonstrate the above steps.

Textual output produced with the DISPLAY command is just a temporary addition to the graphic report. Upon entering a new DISPLAY <graph type> command, all displayed text will disappear. The display can be refreshed, however, by entering the command DISPLAY (see section 3.3.4.3).

Textual information displayed with the DISPLAY TEXT command may be removed from the display with the command:

DELETE TEXT NAMED <name> .

When the command is used, all occurrences of that text will be eliminated. The current display can then be refreshed with the command DISPLAY.

3.3.4.3 Graph Refresh Capability. Once a graph has been displayed the user may wish to alter it. This includes changing colors, modifying the TITLE, adding TIC marks, adding SYMBOLS, specifying scaling parameters, etc. The graph can then be redrawn with these new specifications by entering the GIPSY refresh command:

DISPLAY

or

D.

This command redraws the graph without having to retype the entire graph display command. The refresh command is not allowed following the BUILD NEW REPORT, ACCESS, APPEND, REMOVE, LIMIT, ACCUMULATE, DIFFERENCE or TRANSFER commands.

3.3.4.4 Turning the Graph Off. The user may desire to display a screen with only text, symbols, title and/or classification without a corresponding graph. This may be done using the command:

SET DISPLAY { ON
OFF } .

TEXT TABLE EXAMPLE

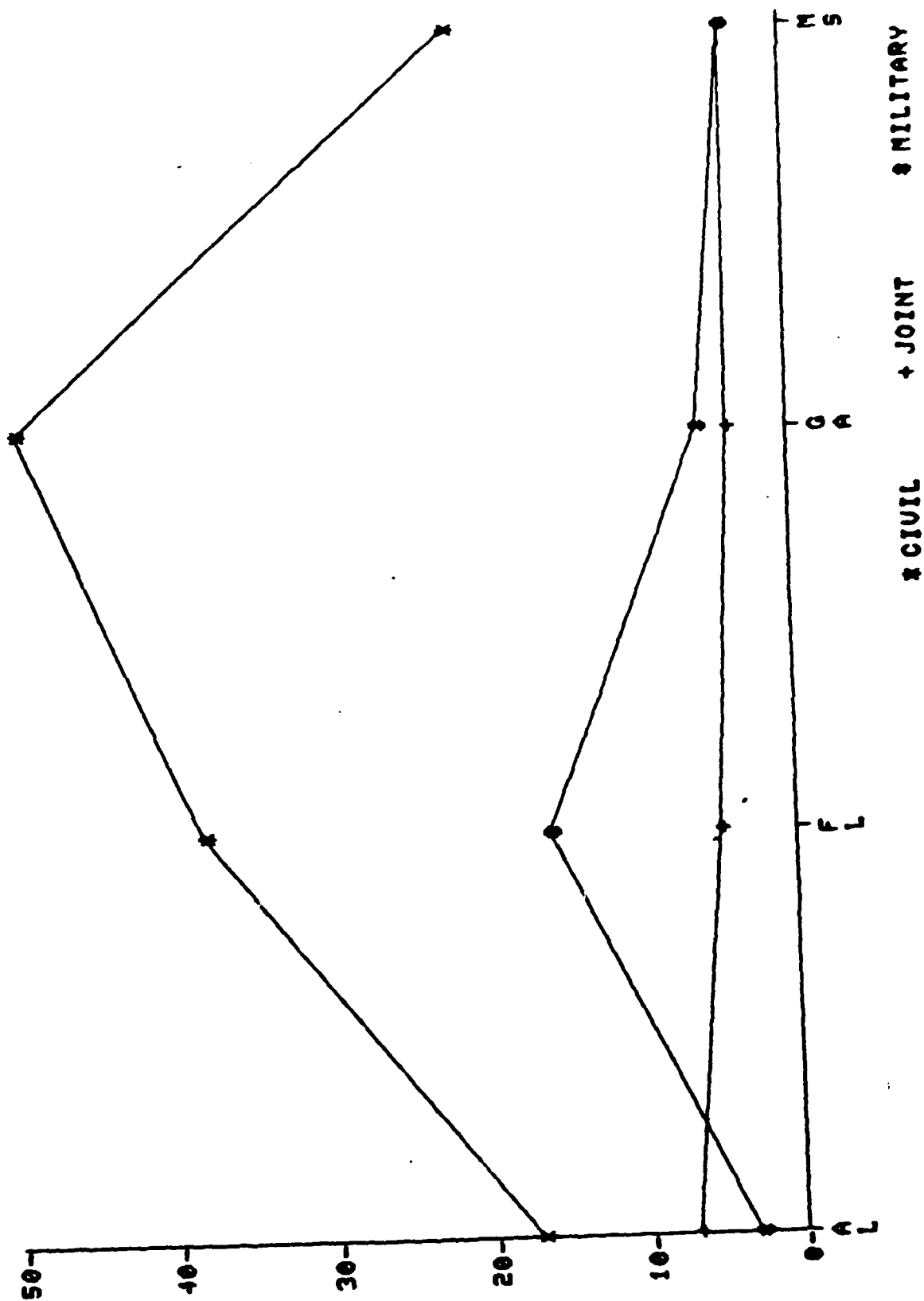


Figure 3-28. Initial Graphic Display

TEXT TABLE EXAMPLE

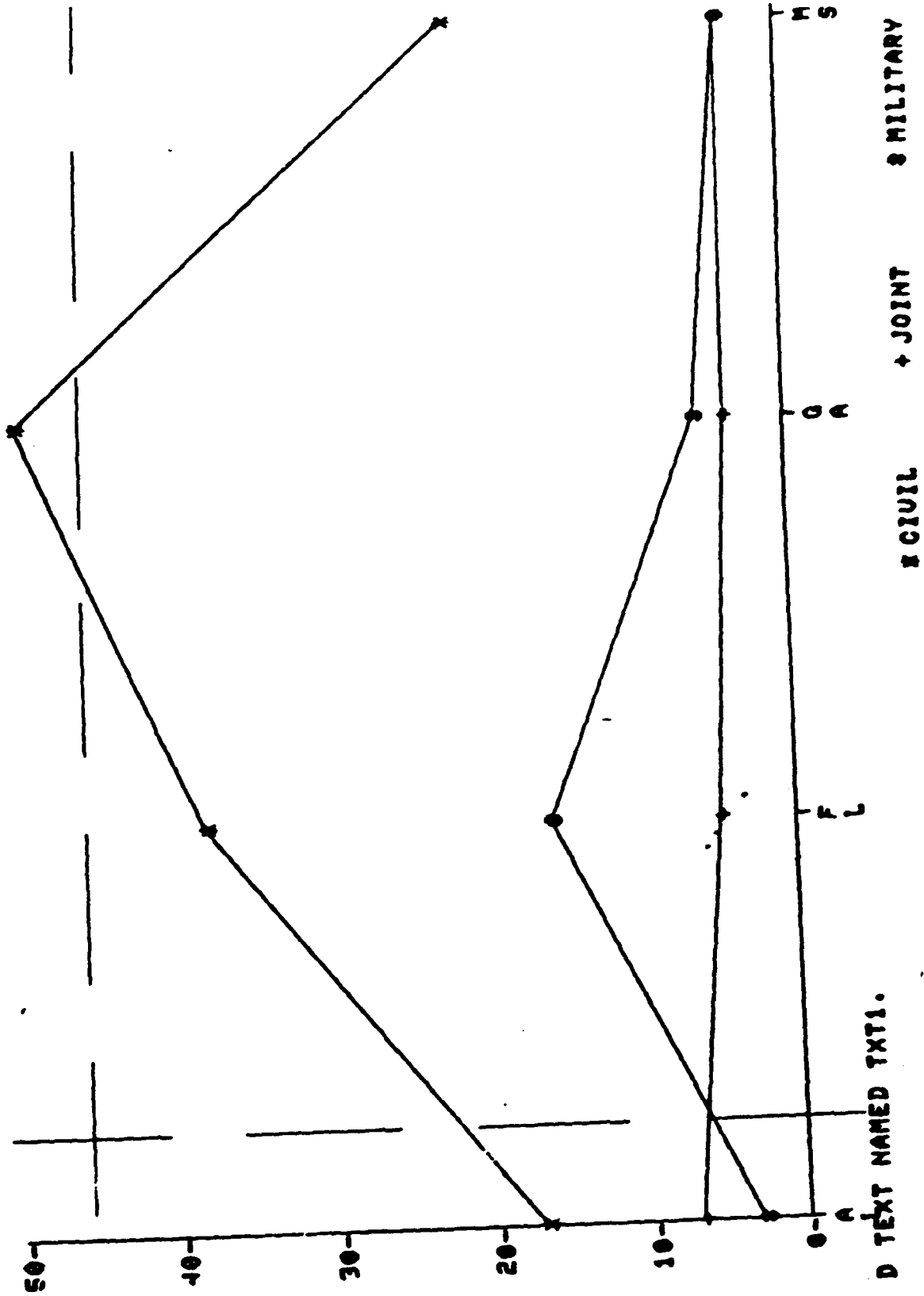


Figure 3-29. Positioning TXT1 With Graphic Cursor

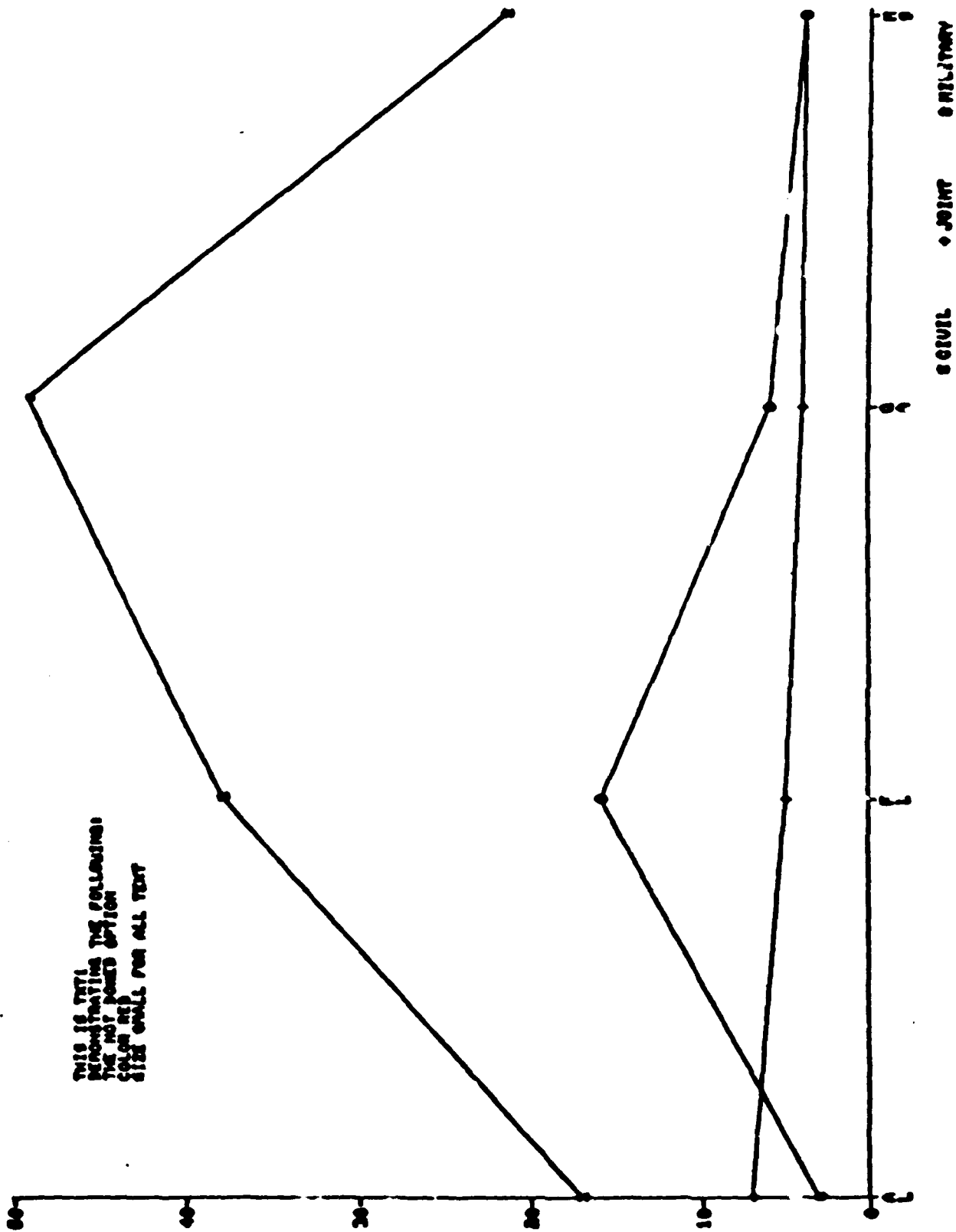


Figure 3-30. Graph With TXT1

TEXT TABLE EXAMPLE

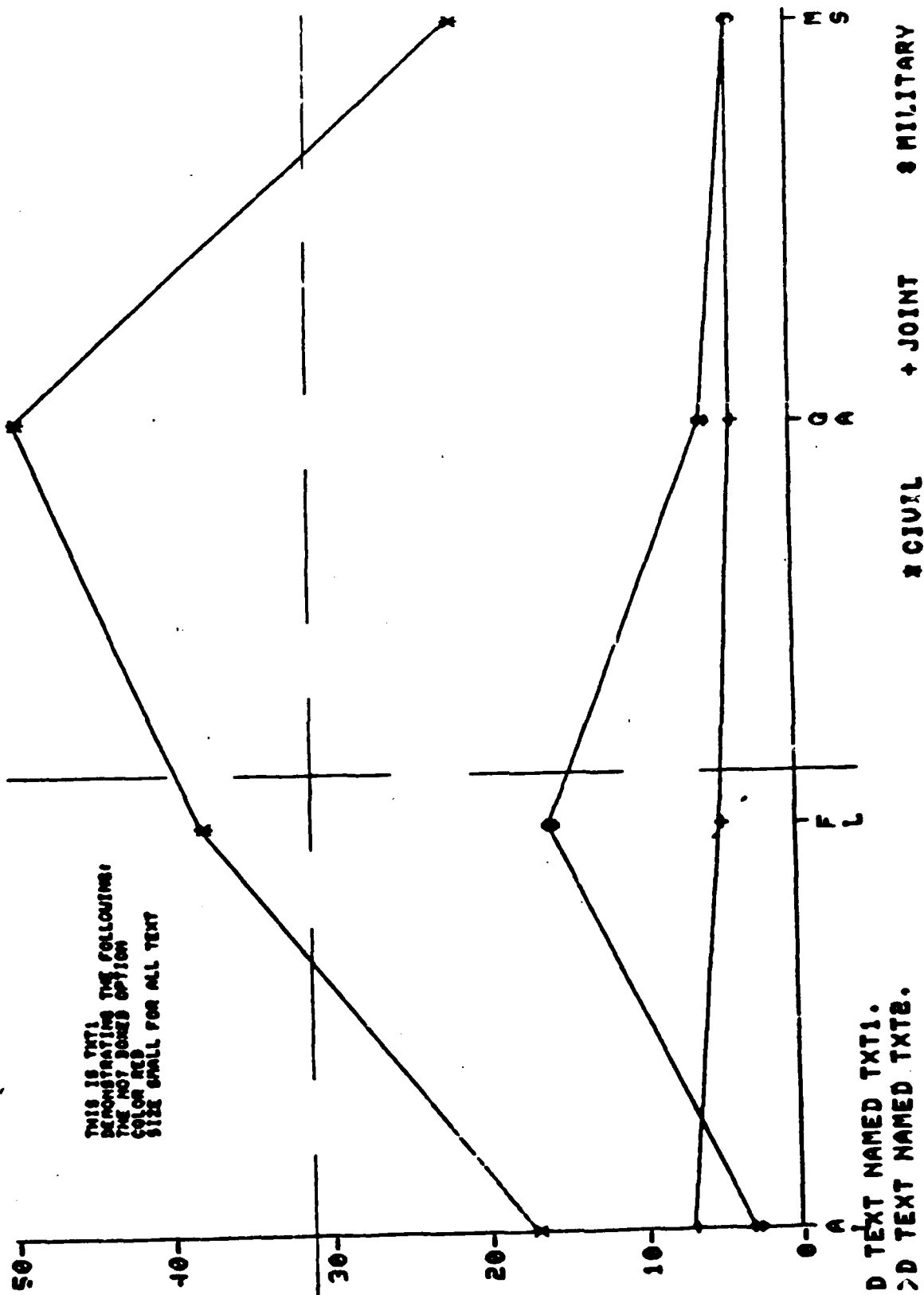


Figure 3-31. Positioning TXT2 With Graphic Cursor

TEXT TABLE EXAMPLE

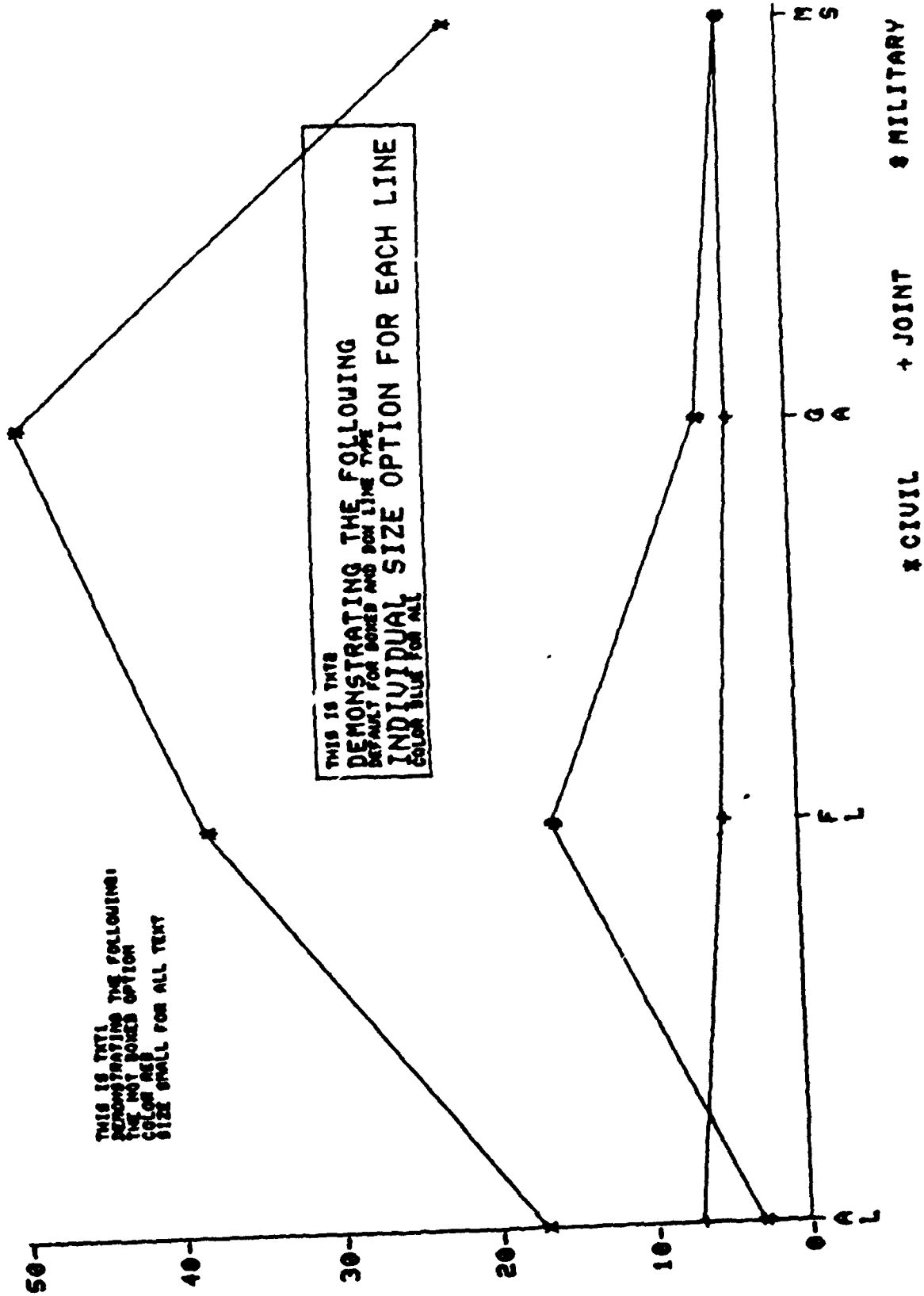


Figure 3-32. Refreshed Graph With TXT1 and TXT2

If DISPLAY is set OFF, subsequent DISPLAY commands will clear the screen and display existing classification and title lines as well as generated symbols and text. Additional symbols and text may then be added to the display.

3.3.5 Saving Statistical Reports. The tabular report (matrix) can be saved from within GIPSY onto a permanent file. Since the tabular report is the basis for all other statistical reports, the user needs simply to access this saved report at a later date in order to produce graphic reports.

The tabular report may be saved after the RUN statement by using the command:

```
{ SAVE      }  
  RESAVE    }      REPORT [ON] <cat/file string>
```

The current tabular report is saved to the specified file as a Graphic Data Set (GDS).

3.3.6 Accessing Saved Reports. GIPSY provides the capability to load a tabular report, either created by GIPSY in a previous run, or created by a user application program. The report must have been saved on a file that GIPSY references as a GDS. GIPSY is directed to access that file with the ACCESS command. This command is used to access either a new GDS or a specific report from the current GDS. The syntax for the command is:

```
ACCESS      { GDS  cat-file string  
              REPORT [<report number>]  
              NEXT }
```

ACCESS GDS <cat-file string> will cause GIPSY to release its access to the current GDS, if there is one, attach the new specified GDS, and load the first report on that GDS into memory. ACCESS REPORT will cause the specified report number to be loaded into memory overlaying the current report that is in memory. If <report number> is not specified, the last report referenced will be loaded. This comes in handy if you have destroyed part of the data or if you simply want to get back to the starting point. ACCESS NEXT simply loads the next sequential report from a multiple report GDS.

3.3.7 Building a New Report. As previously mentioned, GIPSY provides a very powerful capability for building, displaying, modifying and enhancing statistical reports. GIPSY also provides an entire set of commands for building a new report using or modifying the current report. This feature begins with the command BUILD NEW REPORT and is terminated with an END NEW REPORT or simply an END.

The build new report capability would be used to modify the current report for such things as to convert a record count tabulation into percentages, apply weighing factors, normalize variables and units of measure, compute differences and composites, etc. Also, an entirely new report may be built and displayed without referencing a data base.

The language for this capability is necessarily a bit complex; however, information messages are generously distributed throughout the block structure. They will not appear, however, unless requested by entering a carriage return when the console is open for any input other than an error correction. The block structure is initiated by BUILD NEW REPORT and terminated by END NEW REPORT or END.

Specifically:

```
BUILD NEW REPORT.  
  <vector statements>  
  .  
  .  
END NEW REPORT.
```

The vector statements define the following capabilities:

- o ASSIGN - assign new rows, columns, sections, or categories to the current tabular report.

assign new values to an already existing vector or element.

assign new values across the entire tabular report based on conditional computations using the current tabular report.
- o DELETE - delete rows or columns from the report.
- o RENAME - rename the rows and columns of the report.
- o SUBSET - subset the report and rebuild it.
- o CHANGE - change specific entries in the matrix part of the report.
- o DEFINE - deletes current vector mode and creates new vector names for specified vector mode and manually inputs the data values for a tabular report.
- o ADD - appends new vectors to the end of the vectors already in the tabular report.
- o INPUT - supplies data to vectors added via a DEFINE and/or ADD statement or to change the contents of pre-existing vectors.

The LIMIT statement can also be used within the BUILD NEW REPORT block structure. Caution - any LIMIT action is global. Therefore, limits set inside BUILD NEW REPORT will remain in effect even after ending BUILD NEW REPORT.

Whenever a new report function (ASSIGN, SUBSET, CHANGE, etc.) is completed, the one-word statement "REVIEW." may be entered to obtain a display of the intermediate results of modifications to the current report. You may then continue with additional BUILD NEW REPORT functions. REVIEW is essentially a DISPLAY REPORT that does not require you to leave the BUILD NEW REPORT block structure. All options available to DISPLAY REPORT are also available to REVIEW.

3.3.7.1 ASSIGN Function. The ASSIGN statement is used to conditionally, or unconditionally, calculate new sets of values for entirely new rows or columns and add them to the report or to recalculate the set of values for a vector already in the report.

The general format of the ASSIGN block structure is:

```
ASSIGN  [ <vector mode> ]  
        ELEMENT  
  
        [<vector name> = <arithmetic expression>. ]  
        [<new vector name> = <arithmetic expression>. ]  
        [ ELEMENT = <arithmetic expression>. ]  
  
END     [ ASSIGN  
        NEW REPORT ]
```

Using the <vector mode> with the ASSIGN specifies the type of vector (ROW, COLUMN, CATEGORY, or SECTION) which is to be assigned. You can mix the vector modes by respecifying the vector mode prior to assigning a vector value.

ELEMENT qualified the entire data portion of the matrix for modification. ELEMENT has a special meaning, so its discussion will be deferred until after we discuss the assignments for the vectors.

The assignment of new values to an already available vector typically consists of an arithmetic expression to assign data values. A conditional expression can be optionally appended to control when the assignment occurs.

If a new vector name is specified, the vector name will be assigned to the vector (ROW or COLUMN) being created. It may consist of any combination of up to 36 alphanumeric or special characters. If the name includes a special character it must be enclosed in quotes. Remember, for each vector mode assigned the associated vector function will occur. For example, if you are assigning a column, the column function is repeated for every row in the report.

The <arithmetic expression> consists of vector names connected by any of the arithmetic operators, specifically the plus sign "+" the minus sign "-" a comma "," the multiplication sign "*" the division sign "/" an open paren "(" and a close paren ")".

An ASSIGN END or a simple END statement will terminate the ASSIGN operations but keep you in the BUILD NEW REPORT block structure. END NEW REPORT will terminate both the ASSIGN and the BUILD NEW REPORT block structure.

Let's take a moment for a few examples. Given the previously built tabular report of project strength by service, let's add a new project, D which has resources that are 10 percent of the sum of all projects, then redistribute 30 percent of C's resources to project A and add the equivalent of 20 percent of A's adjusted resources to project B:

	A	B	C
ARMY	300	280	800
NAVY	200	400	400
USAF	400	640	200
USMC	600	290	100
USCG	150	300	300

The input commands would be:

```

BUILD NEW REPORT.
  ASSIGN COLUMNS.
    D = 0.10*(A + B +C ).
    A = A + 0.3 * C.
    B = B + A * 0.20.
    C = C - 0.3*C.
  END NEW REPORT.
  DISPLAY REPORT.

```

	A	B	C	D
ARMY	540	388	560	138
NAVY	320	464	280	100
USAF	460	732	140	124
USMC	630	416	70	99
USCG	240	348	210	75

Observe that new column D was added to the end of the report and the modified ones remained where they were. Note also that each statement causes the matrix to be updated immediately, as evidenced by an updated A being used in the calculation of B.

The above illustrations show unconditional vector assignments. Let's attach a condition to one of the assignments. Additionally, let's reduce any service which has more than 500 personnel on project A to 500 and place the excess in a new column called OVERHEAD:

```

BUILD NEW REPORT.  ASSIGN COLUMN.
  "OVERHEAD" = A - 500; IF A GT 500.
  A = 500; IF A GT 500.
ROWS.
  TOTAL = ARMY+NAVY+USAF+USMC+USCG.

```

COL. TOTAL = A + B + C + D + OVERHEAD.
END NEW REPORT.

1 OF 1

	A	B	C	D	OVERHEAD	TOTAL
ARMY	500	388	560	138	40	1488
NAVY	320	464	280	100	0	1064
USAF	460	732	140	124	0	1332
USMC	500	416	70	99	130	1116
USCG	240	348	210	75	0	798
TOTAL	2020	2348	1260	536	170	5798

We threw in a row total to show how to switch back and forth between row and column mode.

A word of caution is in order here; if the vector name (row or column name) is numeric it must be enclosed in quotes; otherwise, GIPSY will misinterpret it as a number. Accordingly, if the vector header contains special characters, including blanks, it must be enclosed in quotes.

The foregoing discussions were limited to vector mode. The last mode is ELEMENT. When in this mode every element in the matrix is available for modification, and vector names have no meaning, only numbers and the descriptor ELEMENT may be used in the arithmetic expression and conditional expression. For example, if we wanted to reduce every element greater than 450 by 10 percent we would specify:

```
BUILD NEW REPORT.  
ASSIGN ELEMENT.  
ELEMENT = ELEMENT - 0.10*ELEMENT; IF ELEMENT GT 450.  
END ASSIGN.  
END.
```

1 OF 1

	A	B	C
ARMY	300	280	720
NAVY	200	400	400
USAF	400	576	200
USMC	540	290	100
USCG	150	300	300

The END ASSIGN terminates the assignment operation but leaves you in the BUILD NEW REPORT block to enter additional vector statements.

We mentioned earlier that the LIMIT statement can be used inside the BUILD NEW REPORT block structure. The following example illustrates the use of the

LIMIT statement in conjunction with the ASSIGN statement using the current tabular report:

1 OF 1

	A	B	C
ARMY	300	280	800
NAVY	200	400	400
USAF	400	640	200
USMC	600	290	100
USCG	150	300	300

Input commands would be:

```

BNR.
LIMIT COLS * IF ANY ELEMENT GE 600 AND ELEMENT LE 750.
ASSIGN ROW.
TOTAL = ARMY + NAVY + USAF + USMC + USCG
END ASSIGN.
LIMIT COLS TO ALL.
END.
D REPORT.

```

1 OF 1

	A	B	C
ARMY	300	280	800
NAVY	200	400	400
USAF	400	640	200
USMC	600	290	100
USCG	150	300	300
TOTAL	1650	1910	1800

The LIMIT statement must precede the associated ASSIGN statement, because processing of an ASSIGN statement occurs immediately upon its entry. This requires that the vector to which the arithmetic result is added must be identified prior to the arithmetic operation.

3.3.7.2 DELETE Function. The DELETE statement is used to conditionally or unconditionally eliminate rows and/or columns from the report. The general form of the statement is:

DELETE $\left\{ \begin{array}{l} \text{ROW} \\ \text{COL} \\ \text{CATEGORY} \\ \text{SECTION} \end{array} \right\}$ <vector name list> [;IF $\left\{ \begin{array}{l} \text{ANY} \\ \text{ALL} \end{array} \right\}$ <element condition>].

The <vector name list> can occur as a list of individual <vector names> separated by commas, ranges of names, or combinations thereof; specifically:

<vector name 1>, [<vector name 2> [THRU <vector name n>]]
[, <vector names>]

If a range of <vector names> is specified, it is interpreted as an inclusive range, left to right, in order of appearance in the report.

DELETE causes an immediate and permanent removal of the identified vector from the report. The conditions which may be attached to the deletion of a row or column are elemental ones. That is, only the special name ELEMENT may be used in the conditional expression. Some examples of the DELETE statement are:

```
BUILD NEW REPORT.  
  DELETE COL A.  
  DELETE ROW USAF IF ALL ELEMENTS GT 100.  
  DELETE COLUMN C IF ANY ELEMENT LT 100.  
END.
```

3.3.7.3 RENAME Function. The RENAME statement is used to assign new vector names either by replacement of the original name or by addition of a prefix or suffix to the existing vector header. The format of the RENAME statement is:

```
RENAME      { ROW  
              COLUMN  
              CATEGORY  
              SECTION } <rename options> .
```

where <rename options> are of the form:

```
<vector name 1> <new name 1> [, <vector names> <new names>] . . .  
  
NAMES <ordered list of names>  
<vector name 1> THRU <vector name n>  PREFIX "<string>"  
SUFFIX "<string>"
```

The <ordered list of names> is a list of new names which will be mapped one-for-one into the THRU sequence of names. Let's use an example to see if we can clarify the rename function using our standard report.

Observe the syntax of the RENAME statement and its effect on the vector names:

```
BNR.  
RENAME ROW ARMY ALPHA.  
RENAME ROW NAVY BETA.  
RENAME ROW USAF GAMMA.  
RENAME ROW USMC DELTA.  
RENAME ROW USCG EPSILON.  
END NEW REPORT.  
D REPORT.
```

1 OF 1

	A	B	C
ALPHA	300	280	800
BETA	200	400	400
GAMMA	400	640	200
DELTA	600	290	100
EPSILON	150	300	300

BNR.
RENAME COLS A THRU C NAMES ABLE, BRAVO, CHARLIE.
RENAME COLS ABLE THRU CHARLIE PREFIX "X-".
END NEW REPORT.
D REPORT.

1 OF 1

	X-ABLE	X-BRAVO	X-CHARLIE
ALPHA	300	280	800
BETA	200	400	400
GAMMA	400	640	200
DELTA	600	290	100
EPSILON	150	300	300

3.3.7.4 SUBSET Function. SUBSET provides the capability to create a subset of the report by cutting vectors or sequences of vectors from the report, reordering and pasting them into a new report. It also provides a capability to calculate new vectors and rename old ones in the subsetting process. SUBSET is different from ASSIGN in that matrix modifications are deferred until SUBSET is complete, while ASSIGN dynamically modifies the report. Like ASSIGN, SUBSET is a block structure bounded by SUBSET...END SUBSET. The specific syntax is:

```
SUBSET <vector mode>.  
    [<vector assignments>.]  
    [<vector mode>. <vector assignments>.]  
END [ SUBSET [ (NULL) ] ] .
```

The <vector mode> defines the ROW, COLUMN or SECTION which is candidate for the SUBSET function.

The format for the <vector assignments> is:

```
INCLUDE <list of vectors to be included in new report>.  
CALCULATE <vector header> = <arithmetic expression>.
```

The <list of vectors to be included in new report> has the same syntax:

```
<vector name> [<alternate name>] [,<vector name> [<alternate name>]] [...]
```

INCLUDE copies the referenced vectors from the current report to the new one and renames them if alternate names are supplied. The INCLUDE vectors are placed in the new report in the order in which they were referenced in the SUBSET structure. New vector names are not available for use until the SUBSET block structure is ended. It is then that the defined subset action actually takes place.

The CALCULATE option is used to calculate new values for the vectors from the current matrix. The operand(s) in the arithmetic expression can be vector name(s) or numeric literal(s).

The <arithmetic expression> consists of vector names connected by any of the arithmetic operators, specifically the plus sign "+", the minus sign "-", a comma ",", the multiplication sign "*", the division sign "/", an open paren "(", and a close paren ")".

The END SUBSET with the (NULL) option cancels the SUBSET function currently defined.

The END or END SUBSET terminates SUBSET but leaves you in BUILD NEW REPORT to allow you to continue with additional NEW REPORT operations with the current report modified by the SUBSET action.

Figure 3-33 shows the report before the following SUBSET commands:

```
BUILD NEW REPORT.
SUBSET COLS.
    INCLUDE AUTHORIZED, ACTUAL, ENLISTED.
    CALCULATE PCT-ENLISTED = ENLISTED/ACTUAL*100.
    INCLUDE OFFICER.
    CALCULATE PCT-OFFICER = OFFICER/ACTUAL*100.
ROWS. INCLUDE AC07 THRU AC03,AD08 THRU AF19,AG23.
END SUBSET.
END NEW REPORT.
DISPLAY REPORT(COMPRESSED).
```

Figure 3-34 shows the results after the above NEW REPORT functions.

3.3.7.5 CHANGE Function. The CHANGE statement is used to manipulate specific matrix elements. The matrix location (i.e., row and column intersection) of the element is specified by naming the intersecting vectors according to the following statement format:

```
CHANGE ROW, COL [ [, CAT ] [, SEC ] .
<vector specifications> = <arithmetic expression>.
.
.
END.
```


UNCLASSIFIED

1 OF 1

PERSONNEL STRENGTH PROFILE BY ASSIGNED MISSION
BASED ON AUTHORIZED AND ASSIGNED STRENGTHS
 (CIPSY SYSTEM TEST 81)

	AUTHORIZED	ACTUAL	OFFICER	ENLISTED	STR-RATIO	MSN-READY
AA01	48	45	5	40	.9375000	3.2000000
AB02	40	31	1	30	.7750000	3.8709678
AC03	65	75	25	50	1.1538462	2.6000000
AC04	65	29	0	29	.4461538	6.7241380
AC05	75	55	12	42	.7333333	4.0909091
AC06	113	73	5	68	.6460177	3.0958904
AC07	15	23	4	19	1.5333333	.6521739
AD08	23	22	21	1	.9565217	5.2272728
AD09	35	35	0	35	1.0000000	4.0000000
AD10	40	52	6	46	1.3000000	2.3076923
AD11	50	55	1	54	1.1000000	1.8181818
AD12	105	103	3	100	.9809524	1.0194175
AE13	29	13	1	12	.4482759	6.6923077
AE14	39	29	13	16	.7435897	6.7241380
AE15	6	7	7	0	1.1666667	4.2857143
AE16	29	24	22	2	.8275862	6.0416667
AE17	49	42	10	32	.8571429	5.8333333
AF18	150	135	1	134	.9000000	4.4444444
AF19	133	100	10	90	.7518797	6.6500000
AF20	170	168	4	164	.9882353	5.0595238
AF21	50	60	25	35	1.2000000	4.1666667
AF22	88	95	5	90	1.0795455	4.6315789
AG23	115	115	12	103	1.0000000	5.0000000

UNCLASSIFIED

Figure 3-33. Tabular Report Input to Subset

UNCLASSIFIED

1 OF 1

PERSONNEL STRENGTH PROFILE BY ASSIGNED MISSION
BASED ON AUTHORIZED AND ASSIGNED STRENGTHS
(GPOV SYSTEM TEST 81)

	AUTHORIZED	ACTUAL	ENLISTED	ENLISTED	PCT- OFFICER	PCT- OFFICER
AC07	15	23	19	82.60870	4	17.39130
AC06	113	73	68	93.15069	5	6.84932
AC05	75	55	42	76.36364	13	23.63636
AC04	65	29	29	100.00000	0	0.00000
AC03	65	75	50	66.66667	25	33.33333
AD08	23	22	1	4.54545	21	95.45455
AD09	35	35	35	100.00000	0	0.00000
AD10	40	52	46	88.46154	6	11.53846
AD11	50	55	54	98.18182	1	1.81818
AD12	105	103	100	97.08738	3	2.91262
AE13	29	13	12	92.30769	1	7.69231
AE14	39	29	16	55.17241	13	44.82759
AE15	6	7	0	0.00000	7	100.00000
AE16	29	24	2	8.33333	22	91.66667
AE17	49	42	32	76.19048	10	23.80952
AF18	150	135	134	92.25926	1	.74074
AF19	133	100	90	90.00000	10	10.00000
AG23	115	115	107	89.56522	12	10.43478

UNCLASSIFIED

Figure 3-34. New Report After Subset

The <vector specifications> are in the format:

<vector name 1>, <vector name 2> [[, <vector name 3>] [, <vector name 4>,]

They may be arranged in any order desired. This specifies the order in which the vector names will be changed. If the matrix is larger than two dimensions, categories and/or sections must also be specified. The <vector name> identifies the intersection of vectors containing the data element to be altered.

The system-defined variable ELEMENT may be used in the arithmetic expression. In this context ELEMENT will contain the current value of the matrix element being referenced.

In the current report:

1 OF 1

	A	B	C
ARMY	300	280	720
NAVY	200	400	400
USAF	400	576	200
USMC	540	290	100
USCG	150	300	300

The following CHANGE statements will alter the values of elements at matrix locations (USMC,A), (USAF,C), and (NAVY,C).

BNR.
CHANGE ROW, COL.
USMC, A = 34 + .10 * 80.
USAF, C = 24 + .10 * 80.
NAVY, C = 14 + .10 * 80.
END.
END NEW REPORT.
D REPORT.

1 OF 1

	A	B	C
ARMY	300	280	800
NAVY	200	400	22
USAF	400	640	32
USMC	42	290	100
USCG	150	300	300

The following CHANGE statement will alter the values of elements at matrix locations (ARMY,A) and (NAVY,A):

BNR.
CHANGE ROW, COL.

```

ARMY, A = ELEMENT * .10.
NAVY, A = 5 - ELEMENT + 20.
END.
END NEW REPORT.
D REPORT.

```

1 OF 1

	A	B	C
ARMY	30	280	800
NAVY	-175	400	22
USAF	400	640	32
USMC	42	290	100
USCG	150	300	200

3.3.7.6 DEFINE Function. The DEFINE statement is normally used to manually create a new matrix (tabular report) when the current set of data is no longer required or there does not yet exist a current report. The syntax is:

```

DEFINE { vector mode <NAMED> list of new vector names }
       { number of vector names <vector mode> }

```

The <vector mode> is either a ROW, COLUMN, CATEGORY or SECTION. The <list of new vector names> is a series of names to be used as the vector(s) being created. Each name must be separated from the other with a comma. If the name contains any special characters, it must be enclosed in quotes.

If the second option is chosen, GIPSY will automatically generate the names. The names GIPSY will generate consist of the vector mode specified along with a sequentially incremented integer. Duplicate labels will not be generated. If a name which would be generated already exists, GIPSY will skip over it. The <number of vector names> is an integer constant defining how many vectors are to be generated. The <vector mode> defines which of the four vector modes the request applies to.

If we wanted to define a new report to GIPSY the following statements would be made:

```

BUILD NEW REPORT.
  DEFINE ROWS NAMED ALPHA, BETA, DELTA, GAMMA.
  DEFINE 5 COLUMNS.
END.

```

The results at this point would be:

	COL1	COL2	COL3	COL4	COL5
ALPHA	0	0	0	0	0
BETA	0	0	0	0	0
DELTA	0	0	0	0	0
GAMMA	0	0	0	0	0

The first DEFINE statement initiates a matrix and creates the four rows indicated. The second DEFINE creates the five columns as shown in the above report. Caution - if you are working with a current report, the first DEFINE statement will zero out the entire matrix and then replace the rows and columns as described in the DEFINE statements.

3.3.7.7 ADD Function. The ADD statement should be used if any portion of the data in an existing tabular report is to be retained. It functions identically to the DEFINE statement except as noted below. The syntax is:

```
ADD      { vector mode <NAMED> list of new vector names }
          { number of vector names <vector mode>           }
```

The following statements would be used to build a report:

```
BUILD NEW REPORT.
  ADD 4 ROWS.
  ADD COLUMNS NAMED AUTHORIZED, ACTUAL.
END NEW REPORT.
```

	AUTHORIZED	ACTUAL
ROW1	0	0
ROW2	0	0
ROW3	0	0
ROW4	0	0

The ADD statement is not destructive to the current report as the DEFINE statement is. The ADD will cause the defined vectors to be appended to the end of the vectors already in the tabular report.

3.3.7.8 INPUT Function. The INPUT statement may be used to supply data to vectors added via a DEFINE or ADD statement or to change the contents of pre-existing vectors. Data may be supplied either as a single statement or in response to detailed prompted input requests. The syntax for the INPUT statement is:

```
INPUT { <vector mode>
        <vector list>;
        FOR <vector mode><vector lists>; } VALUES[.] { <value list>
                                                         ARE ALL <number>
                                                         ARE FROM <number> BY
                                                         <number> }
```

The <vector mode> defines the mode (ROW, COLUMN, CATEGORY or SECTION) to which the new data values are to be supplied.

The <vector list> is a list of vector names belonging to the identified vector mode. Each name must be separated from the other by a comma. A sequential set of names may be included by use of THRU as defined.

The FOR <vector mode> <vector list>; dictates the order in which the data is to be specified. It may also be used to limit the scope of the input

sequence. The INPUT statement may contain up to three vector modes -- one for each vector mode not already used in the statement. The semicolon is required to terminate the vector list. The default order of input request is COLUMN, ROW, CATEGORY, and section. The FOR clause is used to alter this ordering of data to permit the data to be entered in a manner more suitable to the user's data requirements.

The command VALUE establishes the start of data to be input as values. If the end of the statement (period) occurs immediately after the keyword VALUES, then GIPSY will enter a detailed prompt mode which will ask for the information by vector. The prompts for input will continue until the all vectors have been filled. The period is used as a terminator for the prompted input. The report will be considered complete when a period is encountered. Do not attempt to use a period at the end of each prompted input--it will terminate the input process. The input values must be specified according to the vector sequence requested; each value must be separated from the other with a comma. The following example illustrates these features (user-specified input is preceded by the prompt character ">"):

```
BUILD NEW REPORT.
DEFINE ROWS NAMED R1, R2, R3.
DEFINE COLS NAMED A, B, C, D.
INPUT ROW VALUES.
INPUT 4 VALUES FOR ROW R1.
1, 2, 3, 4
INPUT 4 VALUES FOR ROW R2.
5, 6, 7, 8
INPUT 4 VALUES FOR ROW R3.
9, 0, 0, 12.
END NEW REPORT.
DR.
```

Note that when all entries in the report have been entered, GIPSY responds with a standard prompt character ">" rather than input requests prompting. The resulting report is:

	A	B	C	D
R1	1	2	3	4
R2	5	6	7	8
R3	9	0	0	12

If less than a complete vector is to be supplied and the original values in the vector are to be retained, the CHANGE statement (section 3.3.7.5) should be used.

The user who is familiar with the storage of data within multi-dimensional arrays may supply the entire tabular report as the <value list> in the INPUT statement. The <value list>, if supplied, must consist of a series of numbers (separated by commas) in the order the data is to be placed in the tabular report. The default order is to first fill up the column, by supplying a

value for each row, then by category followed by section.

For example, the statements:

```
BNR.  
DEFINE COLS NAMED A,B,C.  
DEFINE ROWS NAMED R1, R2.  
DEFINE CATEGORY NAME I, II, III.  
DEFINE SECTIONS NAMED S-I, S-II.  
INPUT VALUES ARE 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
17,18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,  
35, 36.  
REVIEW.
```

would result in the report:

		I			II			III		
S-I	A	B	C	A	B	C	A	B	C	
R1	1	3	5	7	9	11	13	15	17	
R2	2	4	6	8	10	12	14	16	18	

S-II									
R1	19	21	23	25	27	29	31	33	35
R2	20	22	24	26	28	30	32	34	36

Of course the above report could have been entered more easily by changing the INPUT VALUES statement to:

```
INPUT VALUES ARE FROM 1 BY 1.
```

This FROM clause requests GIPSY to load the specified vectors with values starting at the value following the word FROM and incrementing by the value following BY for each succeeding entry in the report. This entry of values in the tabular report is sensitive to the order and selection stipulated in the FOR clauses.

The clause ARE [ALL] <number> causes each entry in the report to be set to the value specified by <number>.

The <value list>, the FROM/BY, and the ALL clauses may be applied to selected vectors or portions of vectors by use of the FOR clause with a vector and/or vector list to limit the application of the generated or specified values.

For example the statement:

```
INPUT COL B; FOR SECTION S-II; FOR CAT II; VALUES 98,99.
```

would change the values 27 and 28 in the report above to 98 and 99 respectively. The statement:

INPUT ROW R1; FOR SECTIONS FOR CATS
FOR COL B; VALUES ARE FROM 4.5 by 0.5.

would change the values 3, 21, 9, 27, 15, and 33 in the report above to 4.5, 5.0, 5.5, 6.0, 6.5, and 7.0 respectively. Incidentally, column B will now be displayed as real numbers rather than as integer as before; this was caused by the inclusion of a non-integer in that column.

1 OF 1

	I			II			III		
S-I	A	B	C	A	B	C	A	B	C
R1	1	4.5	5	7	5.5	11	13	6.5	17
R2	2	4.0	6	8	10.0	12	14	16.0	18
S-II									
R1	19	5.0	23	25	6.0	29	31	7.0	35
R2	20	22.0	24	26	99.0	30	32	34.0	36

3.3.7.9 REVIEW Statement. Whenever a new report function (ASSIGN, SUBSET, CHANGE, etc.) is completed, the one-word statement "REVIEW." may be entered to obtain a display of the intermediate results of modifications to the current report. You may then continue with additional BUILD NEW REPORT functions. REVIEW is essentially a DISPLAY REPORT that does not require you to leave the BUILD NEW REPORT block structure. All options available to DISPLAY REPORT are also available to review.

3.4 Geographic Displays

The geographic capabilities of GIPSY allow one to display a map of any area of the world, and to overlay the map with geographically located symbols, tracks, great circle paths, range circles, and the results of selected geodetic computations. Political boundaries and topographic details are available upon command. Commands also exist to minimize or prevent overprinting important information on the display.

The geographic display may be built using input data from a user's master data base; it may be interactively built using in-line commands; or, it may be built using combinations of both of these methods. Once the display is built it can be manipulated by zooming in or out from any area of the display; it can be further manipulated by adding or removing displayed detail.

The remainder of section 3.4 is divided into several parts. The first part describes those commands common to building a display from a data base and interactively manipulating an already built display (including interactively building a display from keyboard input). The second part describes the commands which are unique to building a geographic display from a user data base. The third part describes the syntax and semantics of those commands

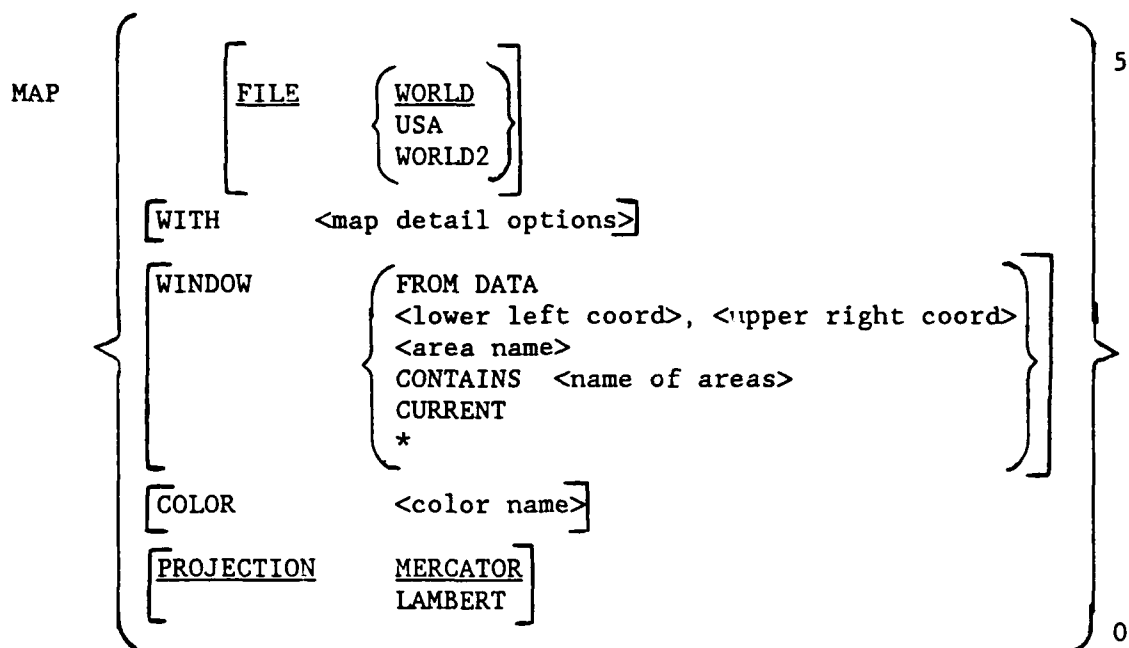
used to interactively manipulate the map and geographic data displays.

If the display is to be built from a user master data base or a formatted subset thereof, the command "GIPSY" should be issued followed by the identification of the data base, the file description, data selection criteria, geographic data processing criteria (symbols, tracks, etc.), title, classification and desired map details. After the processing requirements are specified, the command "RUN" is entered. The file will be queried and the data structure for the requested geographic display will be built. A prompt character will be displayed when GIPSY is ready to accept geographic display commands. (When the commands come from a PCS file, operations will continue without the prompt until the PCS returns control to the keyboard.) If both tabular reports (BUILD TABULAR REPORT) and geographic constructs (SYMBOL or TRACK table) were specified, the management graph/tabular report display mode will be entered. If the input sequence did not include a tabular report, GIPSY will transition to the geographic display processing mode automatically. Transition to the geographic mode is not automatic if tabular reports were built. The transition is then accomplished by entering the command "GEO."

In those cases where the geographic displays are not built from a data base (keyboard display commands, DAFC file, etc.) the geographic display command may be entered directly by using the command GIPSYG (instead of GIPSY).

3.4.1 Geographic Display Definition Statements. All classification, title, and auxiliary control statements (defined in section 3.2.3.) are applicable to all geographic functions, except as noted in the discussion of each command. When building a geographic display there are additional auxiliary control statements which affect only geographic displays. These commands are discussed in the following paragraphs; they may be issued when building a display from a data base and when building a display from the keyboard.

3.4.1.1 Map Definition. The MAP statement is a declarative statement used to identify the map data base you want to use, the geographic window of the area for which data will be qualified, and associated map details. The MAP statement sets up the map parameter for display but does not cause the display to occur. The DISPLAY statement must be used to display the map and associated data. The specified map parameters will be used in all displays unless specifically overridden by another MAP statement. The syntax required to specify the map definition parameters is:



The WITH clause is used to specify the visual, topographic, and political details associated with the actual map background for the display.

The WINDOW clause is used to describe the approximate geographic limits of the area to be included in the geographic display. The limits are specified using coordinates or clear text names which describes the area. The size of the area specified will be adjusted in either the vertical or horizontal direction to insure proper proportionality of the display which always fills the screen area. If no window clause is supplied the entire map will be displayed at the default level of details.

The COLOR clause is used to provide a default color for any map detail for which color is not specifically identified.

The PROJECTION parameter stipulates the map projection on which the entire display is to be based. The default Mercator Cylindrical projection is recommended for most applications. The Lambert Conic Conformal projection is far less efficient and has severe limitations in the areas to which it can be applied.

Each of these clauses will be discussed in detail in subsequent paragraphs.

The GIPSY map display capability is unique in that the apparent resolution of the map is held virtually constant regardless to the size of area to be displayed. GIPSY does not process details which would make no noticeable change in the display. As the area displayed gets smaller finer details are included in the display resulting in an apparent resolution that is independent of the size of the display. The display of the map will occur with the users data when a DISPLAY command is issued. The display of the map itself may be controlled with the SET statement. The command "SET MAP OFF."

will preserve all the MAP statement parameters while inhibiting the actual display of the map. Display status is restored by issuing the command "SET MAP ON."

3.4.1.1.1 The FILE Clause. This clause specifies the map data base is to be used to build the background map. WORLD will cause the display to be produced from a map which contains coasts, islands, major lakes, and international boundaries. It has a moderate level of topographic detail resolution. The maps in figures 2-16 through 2-23 were all produced from the WORLD map file.

USA will cause the map to be displayed from a map file which contains a high level resolution of United States international and state boundaries for the 48 contiguous states as shown in figure 3-35. In figures 3-36 and 3-37, we zoomed in on the Great Lakes area and the Chesapeake Bay area respectively.

WORLD2 specifies an optional map file which is distributed with GIPSY but may not be installed at all sites due to large disk space requirements. It has a very high level resolution and contains many political and topographic details. These details (defined in table 3-1) are included on the display when identified in the WITH clause as <map detail options>. WORLD2 has approximately 60 times as much information as WORLD but it takes only slightly longer to display the additional detail. In some cases, however, a display from WORLD2 is noticeably faster than one with the smaller WORLD map.

Once a map file has been activated it need not be respecified with each issuance of the MAP statement. If no MAP FILE option is specified the last MAP FILE option specified will be used. If not specified the default of WORLD will be used.

3.4.1.1.2 The WITH Clause. This clause is used to specify political or topographic detail and associated display attributes. However, these options are defined only when FILE WORLD2 is used. The <map detail options> are composed of any number of items from table 3-1; each item may have a line type and color associated with it. Specifically, <map detail options> may be defined as:

WITH <detail> [[[COLOR <color>], [LINE TYPE <line type>]]],...

where <detail> is any item selected from table 3-1.

This phrase (and its options in parentheses) may be repeated for each detail selected from table 3-1. If a WITH clause is not specified, NORMAL (from table 3-1) is assumed.

Color, line type, and width specifications are not order-dependent. If more than one option is specified, the clauses should be separated by a comma. As many map details and associated options as desired may be specified; each phrase should be separated by a comma.

The specification color may be replaced by RED, BLUE, GREEN, BLACK or any

USA MAP FILE

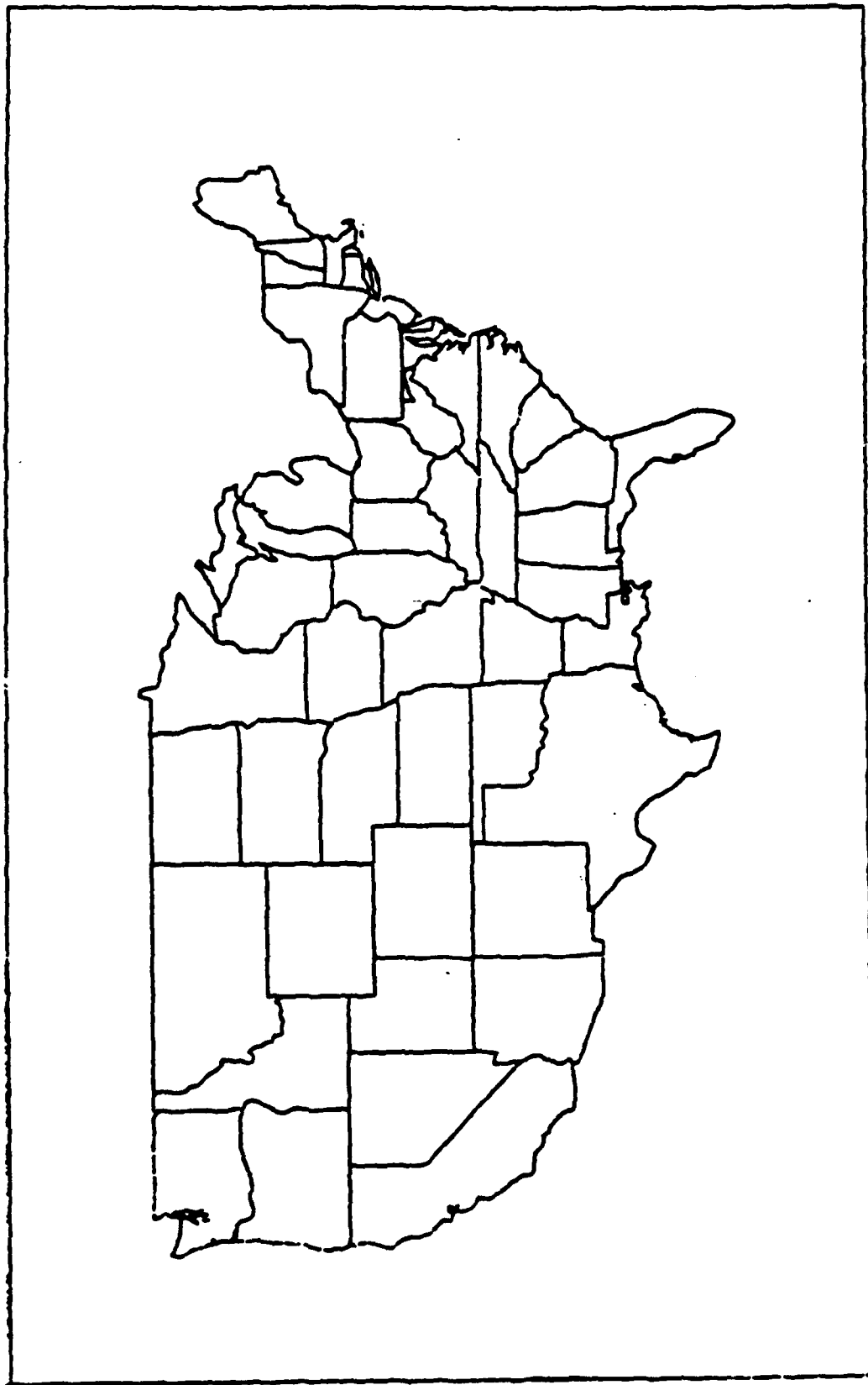


Figure 3-35. USA Map File

USA MAP FILE
GREAT LAKES AREA

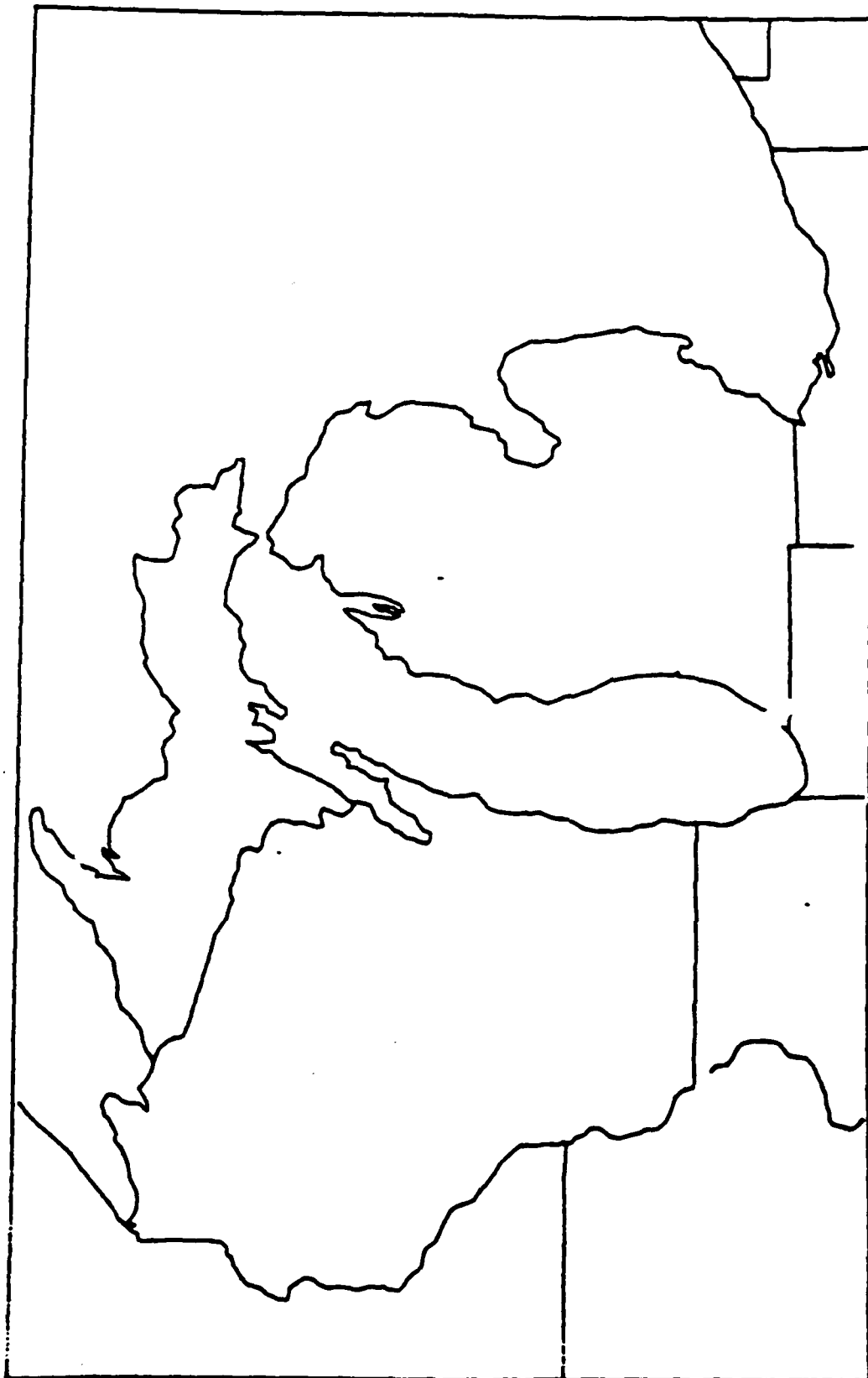


Figure 3-36. Software Zoom on Great Lakes Area From USA Map File

USA MAP FILE
CHESAPEAKE BAY AREA

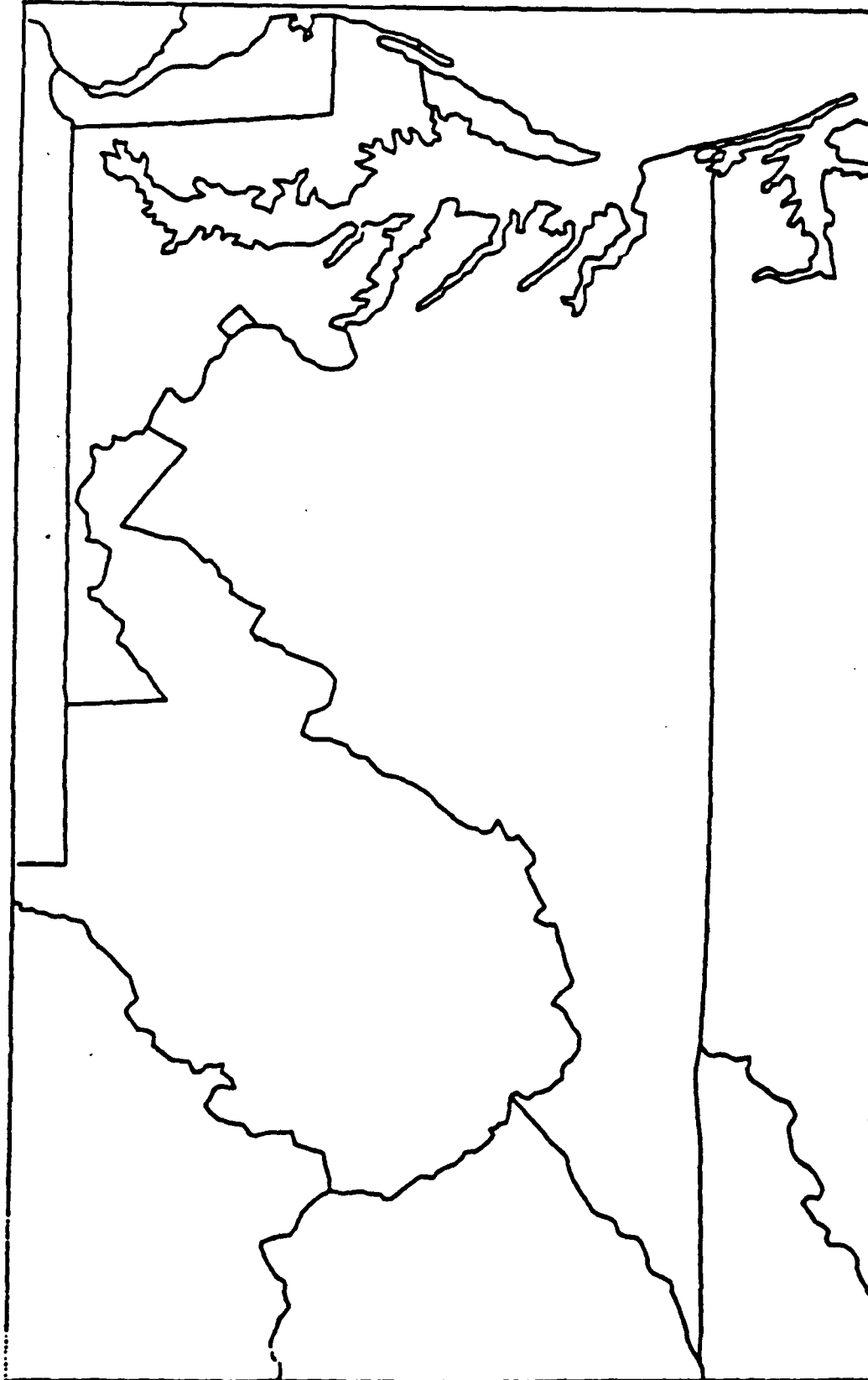


Figure 3-37. Software Zoom on Chesapeake Bay

Table 3-1. WORLD2 Political and Topographic Map Details (Part 1 of 2)

Each major heading option will qualify the subordinates below it. All subordinate options may be selected individually.

ADMIN (administrative boundaries)	RAILROADS
	BROAD GAUGE RAILROADS
	STANDARD GAUGE RAILROADS
	NARROW GAUGE RAILROADS
	UNDIFFERENTIATED RAILROADS
ALL (all map details)	
BOUNDARIES	
COUNTRY BOUNDARIES	REEFS
DELIMITED BOUNDARIES	
DISPUTED BOUNDARIES	RIVERS
INDEFINITE BOUNDARIES	PERMANENT MAJOR RIVERS
OTHER BOUNDARIES	ADDITIONAL MAJOR RIVERS
	ADDITIONAL RIVERS
	MINOR RIVERS
	DOUBLE-LINED RIVERS
	INTERMITTENT MAJOR RIVERS
	INTERMITTENT RIVERS
CANALS	
IRRIGATION CANALS	
LESSER CANALS	
MAJOR CANALS	
COAST	INTERMITTENT RIVERS
	INTERMITTENT MAJOR RIVERS
GLACIERS	INTERMITTENT MINOR RIVERS
	INTERMITTENT RIVERS
ICE SHELVES	
MAJOR ICE SHELVES	MAJOR RIVERS
MINOR ICE SHELVES	PERMANENT MAJOR RIVERS
	ADDITIONAL MAJOR RIVERS
	DOUBLE-LINED RIVERS
	INTERMITTENT MAJOR RIVERS
ISLANDS (see note 1)	
MAJOR ISLANDS	PERMANENT MAJOR RIVERS
MEDIUM ISLANDS	PERMANENT MAJOR RIVERS
MINOR ISLANDS	DOUBLE-LINED RIVERS
LAKES (see note 1)	ROADS (see note 3)
MAJOR LAKES	MAJOR HIGHWAYS
MEDIUM LAKES	MAIN ROADS
MINOR LAKES	GRAVEL ROADS
	UNSURFACED ROADS
	TRAILS
MAJOR FEATURES (See note 2)	
NORMAL	SALT PANS
COASTS	MAJOR SALT PANS
BOUNDARIES	MINOR SALT PANS
	STATES (U.S. State and Canada Province Boundaries)

Table 3-1. WORLD2 Politital and Topographic Map Details (Part 2 of 2)

Note 1: Both islands and lakes will appear if either ISLANDS or LAKES details are specified.

Note 2: MAJOR FEATURES will display all map details except the following: INTERMITTENT RIVERS, INTERMITTENT MINOR RIVERS, MINOR RIVERS, ADDITIONAL RIVERS, GRAVEL ROADS, UNSURFACED ROADS, and TRAILS. Since less details are drawn, maps with "MAJOR FEATURES" will be displayed on the terminal screen much faster than maps with "ALL".

Note 3: Road data is not available and will not be displayed for the following: Greenland, Iceland, Canada, U.S., Mexico, Caribbean Islands, southern Brazil, Bolivia, Paraguay, Uruguay, Chile, Argentina, Finland, Norway, Sweden, Indonesia, Papua-New Guinea, Australia, New Zealand, and islands in the Pacific Ocean.

color defined in the terminal description applied at your site. Color names and their associated definitions are terminal dependent and defined for each device individually. See section 3.2.3.17 for a more detailed discussion of color capabilities.

The <line type> item dictates that the specified detail will be drawn with the selected line type. If not specified, solid lines will be used. Options and the lines produced are:

SOLID	_____
DOTTED
DASHED	-----
DOT-DASH
LDASH	— — — — —

3.4.1.1.3 The WINDOW Clause. The WINDOW clause is used to describe the geographic limits of the area to be included in the display. These limits may be described by specifying actual geographic coordinates to identify the lower left and upper right corner of the area to be displayed.

Alternatively, the limits may be described by specifying the name or list of names of the area to be included in the display, or by allowing GIPSY to figure out what the window should be by looking at the data. The area actually displayed will be automatically extended if needed and scaled to the size of the view surface.

GIPSY will compute the smallest area that includes all your data points if the WINDOW clause is WINDOW FROM DATA.

The "<lower left coord>, <upper right coord>" defines a pair of geographic coordinates in one of the standard coordinate forms. The standard coordinate forms are:

DDHDDDH
DDMMHDDMMH
DDMSSHDDMMSSH

where the D's represent numeric degree digits, the M's represent numeric minute digits, the S's represent numeric second digits, and the H's represent appropriate hemispheres. The leftmost set of letters represents latitude values and the rightmost defines longitude values. The hemisphere associated with the latitude is N for North and S for South; on longitude it is E for East and W for West. This definition of a geographic coordinate is applied throughout the system in the same manner. Any reference to a coordinate accepts these coordinate forms.

The window may be visualized by extending the meridian of the lower left coordinates northward to intersect the parallel of the upper right coordinate; the parallel of the lower left coordinate is extended to the right to

intersect the meridian of the upper right coordinate. The rectangular area contained within these lines will be contained within the display area.

The <area name> option allows a user defined name associated with a coordinate pair or the political name of a geographic area to be used in lieu of specifying geographic coordinates. This name of an area is referred to as an <area name> in the syntax definition of the statement. Appendix H lists the names of all the areas which are predefined by GIPSY. You may specify additional area names or override GIPSY predefined names by defining those names in a window table (discussed in section 3.4.1.2). When using Appendix H, you may use either the WINDOW NAME or the COUNTRY-CODE entry. Any area name which contains special characters or blanks must be enclosed in quotes.

The maps in figures 3-35, 3-36, and 3-37 were generated using the following map statements, respectively:

```
MAP FILE USA, WINDOW 184610N1282700W, 512027N0665529W.  
DISPLAY MAP.  
MAP WINDOW = MICHIGAN.  
DISPLAY MAP.  
MAP WINDOW VIRGINIA, FILE USA.  
DISPLAY MAP.
```

The DISPLAY command shown here will be discussed in detail later. For now, we will limit it to the minimum required to illustrate discussed commands. The first example established the map data base as the USA map data base; subsequent MAP statements do not require a FILE clause since the first statement has established the desired map. Note, however, that the FILE clause may be respecified at any time. The first illustration also shows the use of geographic coordinates to define the window. An <area name> from Appendix H could have been used to produce the same display (i.e., figure 3-35 could have been produced with):

```
MAP FILE USA, WINDOW "UNITED STATES".  
DISPLAY MAP.
```

In some cases it is convenient to describe the window by listing the names of areas and locations which are contained within the desired display area. The CONTAINS clause permits this method of describing a window. The <names of areas> in the syntax may be comprised of any combination of area names from Appendix H or location names from Appendix I or any validy defined name which associates a coordinate or pair of coordinates to a name (discussed in sections 3.4.1.2 and 3.4.1.3).

A map of the eastern half of the United States would be produced under the following map definition:

```
MAP WINDOW CONTAINS "KEY WEST", CHICAGO, MAINE, TAMPA.
```

In this case, the extreme parallels and meridians defined by this set of names

would be used to produce the rectangular window which contains these three locations and one area.

Note that location names and area names are mixed in the same statement. In this case, the right edge of the display will be dictated by the easternmost portion of Maine, the left edge of the display by Chicago, the bottom by Key West, and the top by Maine. These limits may be further expanded to produce a display in the required proportions.

If the window is defined as a point (i.e., a <location name> is supplied as the window), a one degree by one degree area around that point will be used as the window.

"WINDOW CURRENT" and "WINDOW *" are special forms which take the geographic limits of the display currently on the screen as the new window. The limits of the display currently on the screen may be different from that in the map definition if the user has zoomed in on a particular area or if the user has temporarily displayed an area different from the MAP WINDOW specification.

3.4.1.1.4 The COLOR Clause. This clause dictates the color to be used for all map details not specifically colored. The <color name> is the same as defined previously in section 3.4.1.1.2.

3.4.1.1.5 The PROJECTION Clause. The Mercator Cylindrical projection is GIPSY's default map projection because the entire world (except for polar regions) can be displayed using this projection. The Lambert Conic Conformal projection also is available; however, the user must assume responsibility to assure that no attempt is made to display a map area inappropriate for a Lambert projection. A complete discussion on map projections is provided in Appendix F.

Observe the results shown in figure 3-38 from the following map statement:

```
MAP FILE WORLD2, WINDOW CONTAINS VIRGINIA, MARYLAND;  
WITH COAST (LINE TYPE DOTTED), STATES (LINE SOLID)  
AND MAJOR RIVERS (LINE DASHED) AND LAKES (LINE DOT).  
DISPLAY MAP.
```

Note that items of a list are separated by commas; lists or clauses are separated by semicolons.

3.4.1.2 Area Names. Names may be assigned to define an area via a block structure called a WINDOW TABLE. The WINDOW TABLE allows the user to assign a pair of coordinates to a user specified name augmenting or overriding Appendix H. The name may be subsequently used in other statements. The syntax for this block structure is:

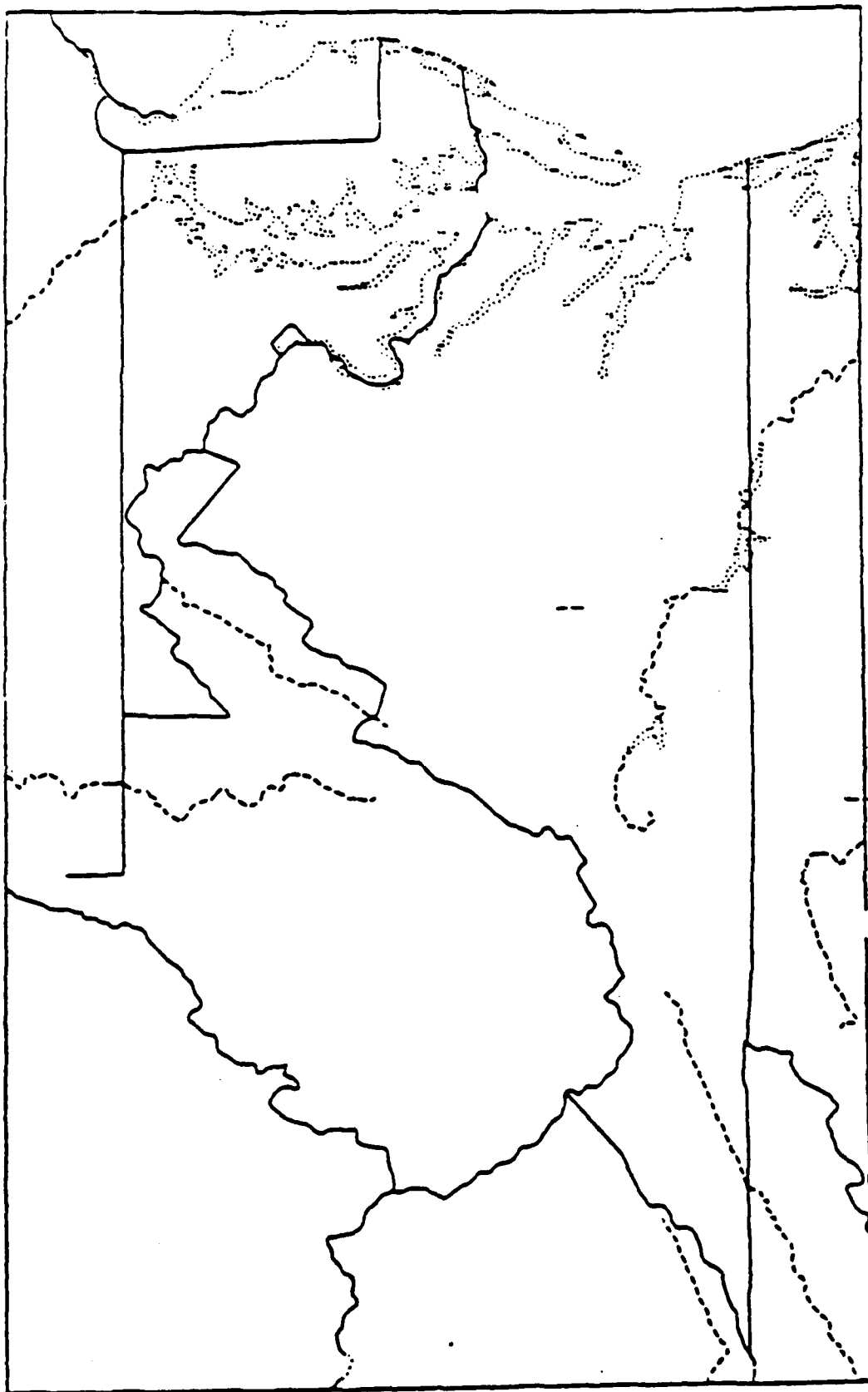


Figure 3-38. Software ZOOM on Virginia and Maryland From WORLD2 Map File

WINDOW TABLE.

<area name> - <lower left coord>, <upper right coord>.

.
.
.

END.

The user generated <area name> may be up to 24 characters in length, the first character of which must be alphabetic. If the name contains any special character (e.g., blank, period, etc.) other than a hyphen it must be enclosed in quotes. It may be enclosed in quotes at any time. There is no limit to the number of elements included in a WINDOW TABLE. However, search time and memory requirements become excessive when the table contains more than 200 to 250 names. The WINDOW TABLE may be saved as a separate PCS file that is brought in as required.

If only one area name is to be defined, an abbreviated form may be used to associate the name with the coordinate of an area. The syntax is:

WINDOW <area name> - <lower left coord>, <upper right coord>.

This statement has the effect of a WINDOW TABLE defining one name.

The WINDOW TABLE or its abbreviated form is not to be confused with the WINDOW clause of the MAP statement. The WINDOW TABLE and its abbreviation merely associate a name with a pair of coordinates and allow that name to be used in place of the coordinates. If the WINDOW TABLE contains a name that is the same as a name in GIPSY's reference file for area names, the WINDOW TABLE name will override the GIPSY name. This condition is assumed to be deliberately generated, hence it does not generate an error message.

Observe the following examples of WINDOW TABLES:

WINDOW TABLE.

ARKANSAS - 330006N0943734W, 363005N0893944W.

MISSOURI - 35N096W, 41N089W.

OKLAHOMA - 3338N10301W, 3700N09426W.

END.

WINDOW "UNITED STATES" - 24N76W, 55N126W.

These names appear in Appendix H; however, the coordinates specified here will be used since any user specified name overrides the default name assignments.

Note that different coordinate formats can be used within a single coordinate table. However, different coordinate formats cannot be mixed within a single definition. Leading zeroes are required when they appear in a coordinate field.

3.4.1.3 Location Names. Names may be assigned to the coordinates of a location via a LOCATION TABLE. The LOCATION TABLE is a block structure directly parallel to the window table discussed in section 3.4.1.2. Names defined in a LOCATION TABLE augment the location names in Appendix I. The syntax for the LOCATION TABLE is:

LOCATION TABLE.

<user generated name> = <geographic coordinate>.

.
.
.

END.

The abbreviated form is:

LOCATION <user generated name> = <geographic coordinate>.

3.4.1.4 Geographic Display Control. There are several features in a geographic display the user may want to control. These factors are controlled by the SET statement. These SET options are discussed here because they are unique to geographic displays. These capabilities augment those discussed earlier in section 3.2.3.5.

The additional SET command syntax is:

- (1) SET { MAP
GRID
PROTECTION } { ON
OFF } .
- (2) SET PROTECTED <display item> { OFF
ON } .
- (3) SET PROTECT AREA { <area name>,
<coordinate pairs>,
CONTAINS { <area names>,
<coordinate pairs> }
CURRENT
*
OFF
ON } .
- (4) SET VISIBLE <resolution factor>.
- (5) SET MAP RATIO { NORMAL
FLOATING
<height> [HIGH] BY <width> [WIDE] } .

SET MAP ON or OFF determines whether the background map will be displayed. Display time can be saved if one sets map display off so that only data will

appear on the display. To cause the map to be reactivated, simply issue:

SET MAP ON.
DISPLAY.

to have map detail restored to the display.

If a grid has been specified, SET GRID OFF prevents the display of the grid on all subsequent displays until a SET GRID ON command has been issued; SET GRID ON allows the normal display of the grid.

GIPSY has the capability to protect portions of the display from being overprinted. This protection (the details of which will be discussed) can be turned on and off with the SET command. The overprint protection is not absolute. The display area will be searched to locate and identify an area which does not contain protected details. If none can be found, the text will be plotted at its original location.

SET PROTECTION ON (or OFF) turns the entire protection capability on (or off).

SET PROTECT display items ON (or OFF) controls the protection of specific elements of the display. The display items which can be specified are discussed below.

MAP requests all map background materials to be protected from first text, then offset lines and list boxes.

LINES request GIPSY to minimize interference caused by offset lines generated by the protection capability (e.g. offset pointers).

TRACKS request that GIPSY minimize interference with user data represented as tracks.

CIRCLES request protection of geographic circles created by a DISPLAY or GENERATE command (to be discussed later) where possible.

SET PROJECT AREA <area options> allows the user to list specific areas of the map to be protected. The <area options> are listed below.

<area names> from the window table may be protected. Any number of areas may be listed, separated by commas or blanks. Only those areas named will be protected, while intervening areas will remain unprotected.

<coordinate pairs> specifying the lower left and upper right corners of an area may also be listed. Each coordinate pair is treated the same as the area names described above.

CONTAINS [<area names> and/or <coordinate pairs>] allows the user to specify a single area to be protected by listing area names and/or coordinate pairs within the protection area. A singly protection area will be defined which

contains all area names/coordinate pairs listed.

CURRENT or * defines the area to be protected as the area which is currently being displayed; i.e., the current map window.

OFF/ON sets protection off or on for all protection areas. If set off, the defined protection areas remain defined but protection will not take place. If SET PROTECT AREA ON is specified, areas previously protected but set off will again be protected.

Examples of usage of these options will be included in the discussions on displaying geographic reports. As mentioned previously, GIPSY adjusts the resolution of the displayed map according to the scale of the window being displayed. With the SET VISIBLE command the user can control this feature. The resolution factor is a numeric value zero or greater. A value of 1 gives the default resolution. Greater than 1 gives less resolution, whereas less than 1 will produce a map with greater detail. Even a resolution factor of 0, however, will not give more detail if the resolution of the terminal is the limiting factor.

The size of the rectangular box which surrounds the geographic display is normally set by GIPSY, the ratio of the height to width being a fixed value. The displayed map will, therefore, contain the specified window along with any area needed to fill the box. If SET MAP RATIO FLOATING is specified, the box will automatically adjust so that only the defined window is displayed. The size of the box can also be fixed by specifying the <height> BY <width> ratio; however, the map will probably be distorted.

3.4.1.5 Geographic Grids. GIPSY provides a mechanism for generating a geographic frame of reference for the display. This frame of reference we call a grid, hence the GRID statement. This statement can cause labeled meridians and parallels to be generated, either automatically or via detailed user specification of the grid lines.

The syntax for the statement is:

```
GRID      [ SIZE <size option>           ] 5  
          [ COLOR <color option>         ]  
          [ LINE  <line option>          ]  
          [ STARTING AT <lower left coordinate> ] 0  
          [ INCREMENT BY <grid intervals> ]
```

If the statement is terminated after the imperative GRID, the system will calculate a grid to insure that at least two meridians and parallels will be displayed within the viewing area. When using this option, the grid lines are automatically recalculated when the viewing area changes or the scale of the display changes.

If STARTING AT <lower left coordinate> is specified, that point will become the origin of a user-specified grid with grid lines stepped off to the right

and up from this point. This coordinate follows the standard coordinate format. The grid increment defines the desired spacing between grid lines. It is specified in degrees and minutes. For example:

GRID STARTING AT 24N125W INCREMENTING BY 0130.

will cause a grid line to be drawn every one and a half degrees from 24N125W to the right edge and top edge of the screen.

Size for the grid labels, line types, and color may be optionally specified on the GRID statement. If not specified, the size and color will default to the current settings and line type will default to a dotted line.

In almost every case it is adequate to simply enter the one word command:

GRID.

3.4.2 Data Base Geographic Displays. The capability to issue statements instructing the system to build a geographic display from a user's pre-existing file is one of the most powerful features of GIPSY. That pre-existing file can be the user's master data base, a subset of a larger file saved from some retrieval or data management process, output from some modeling or simulation process, a card image file, etc. In any case, the FILE statement is used to identify the file; the FDT statement is used to provide a description of the file; and, the classification, title and auxiliary function statements may be used to establish other operating parameters. Then a symbol table or track table from this section may be used in conjunction with the statements in section 3.4.1 to build a geographic display from a data base.

Displays built from the user's data base consist of symbol or track plots possibly augmented by classification, title, grids, etc. Detailed discussions on symbol plots and track plots follow.

3.4.2.1 Symbol Plots. A symbol plot is a geographic display with graphic symbols, text, or geographic circles overlayed on a geographic frame of reference. The user's data file must contain at least the coordinate at which a symbol is to be plotted, and information used as the symbol, or to be used in assigning a symbol to the coordinate location.

GIPSY must be told which field contains the coordinates for a symbol definition. A default field can be established with the following syntax:

SYMBOL COORDINATE - <fieldname>.

where <fieldname> must be a reference to a coordinate type field of any legal length. Any symbol definition in the symbol table which does not define a symbol coordinate field will use this field.

Symbols (including text) to be plotted at a location are provided via a block structure called a SYMBOL TABLE. The SYMBOL TABLE provides the symbols,

display options, and definition of conditions under which the symbol definition is to be acted upon. The body of the SYMBOL TABLE may be made of two classes of symbols -- textual (and enhanced textual) and circles. These classes will be discussed in detail as we discuss the semantics of the SYMBOL TABLE. The syntax for this block structure is:

```
SYMBOL TABLE      [CONTINUED]      [()].

    symbol          []
                   [IF ]
    \CIRCLE RADIUS  <radius> <units>
    .
    END.
```

The block structure declarative, SYMBOL TABLE, may contain a number of options to serve as defaults and symbol display controls. This same set of <symbol options> is available with each symbol definition.

The CONTINUED option allows a previously specified SYMBOL TABLE to be reopened to add new definitions without affecting the existing definition. New <symbol options> on a CONTINUED SYMBOL TABLE only affect the new definitions. The <symbol options> may appear either on the SYMBOL TABLE statement or on the individual definitions.

Any <symbol option> appearing on the SYMBOL TABLE statement applies to all subordinate symbol definitions. If the same or a conflicting option appears on a subordinate symbol definition, the option specified on the subordinate symbol definition takes precedence over that specified on the SYMBOL TABLE statements. In effect, the options specified on the SYMBOL TABLE statement establish the default options for the entire symbol table. The <symbol options> may contain any number of the following:

```
SIZE <size options>
COLOR <color name>
PROTECT
GROUP SEPARATELY
NOT GROUPED
CENTER
MARKER
NAME <symbol entry name>
LINE <line type>
COORDINATE = <fieldname>
BLANKING
```

The options must be enclosed in parentheses if they appear on the SYMBOL TABLE statement. Options should be separated by commas.

SIZE <size option> indicates that symbols are to be displayed using the character size specified by <size option>. JUMBO, LARGE, MEDIUM, SMALL as

previously defined are the valid <size options>.

COLOR <color name> identifies the name of the color to be used in displaying the symbol.

PROTECT requests that the symbols be protected from overprint by other symbols. GIPSY will search for an unused space to write the symbol or text and reserve that space so that no other symbol can overprint it. If no unused space is found the information is displayed at the location defined by its coordinate. Symbols with exactly the same coordinate locations (i.e., collocated symbols) will be collected, protected, and displayed as a single list. A box will be drawn around each list of collocated symbols. When symbols are relocated away from their coordinate locations, lines are drawn from the symbols to the point where the data represented by the symbol should have been plotted. GIPSY will attempt to avoid drawing crossing lines and to avoid crossing through other symbols.

GROUP SEPARATELY will cause collocated symbols from different symbol definitions to be collected in separate lists for protection purposes. The grouping default is to collect all collocated symbols in a single list even though the symbols come from separate symbol definitions.

NOT GROUPED prevents all grouping of symbols, i.e. no lists of collocated symbols are built; each symbol is individually protected if PROTECT is specified.

CENTER dictates that the symbol will be plotted such that the first character is centered over the point representing the symbol's coordinate. If CENTER is not specified, the symbol is plotted such that the lower left corner of the character space represents the location specified by the symbols' coordinate. Don't try to center and protect the same symbol.

MARKER designates that the subject symbol is used as a location marker and is not to be relocated for PROTECT purposes. The MARKER symbols will not be protected from write through by other symbols.

NAME <symbol entry name> allows the user to assign a name to the symbol definition in order to be able to subsequently reference the definition. If not specified the name will default to symbol .

LINE TYPE <line type> directs that relocation lines and list collector lines be drawn with the type of line specified following LINE TYPE. The values for line type are the same as described previously in the MAP statement discussion (section 3.4.1.1).

COORD - <fieldname> allows a temporary override of the SYMBOL COORDINATE statement. This mechanism in effect allows each symbol definition to have a different coordinate field definition.

An unlimited number of symbols may be defined in a symbol table. However,

only one symbol table can be defined. Any subsequent symbol table will totally replace any existing symbol table, except when CONTINUED is specified on the SYMBOL TABLE statement.

The body of the symbol table has two types of symbol definitions -- symbols composed of BCD characters including special GIPSY graphic characters (i.e., symbols) and symbols defined as geographic circles (i.e., circles) with a user specified radius.

In the first form of the symbol definition, <symbol> may be a fieldname whose content will be plotted as a symbol or it may be a literal enclosed in quotes. A literal is limited to a maximum of 12 characters. The content of the field may be up to 132 characters in length. Several special GIPSY graphic characters have been defined to augment the textual character set. The character embedded in a literal " " (string in quotes) is paired with the character following it to identify a special GIPSY graphic character (GGC).

The characters currently defined are:

```
\A + \C ^ \E O \G □  
\B ♦ \D * \F x \H .
```

See section 3.4.2.1.1 for a discussion on how to define your own special graphic characters.

The string \CIRCLE (not in quotes) specifically identifies a geographic circle as the symbol to be plotted. A geographic circle is a circle drawn on the surface of the earth. Geographic circles may not appear as a true plane circle when plotted on the map. Each point on the circle is processed through the projection algorithm of the specified projection. This insures that the area enclosed by the circle represents the area within that defined radius on the surface of the earth. A plane drawn circle does not have that property. Note the variations in the size and shape of the geographic circle in figure 3-39, although they all have the same radius. The appearance of the geographic circle is very much dependent upon the latitude and projection.

The radius of the circle can be stated directly in the statement or extracted from the data base; the <radius> must be an integer value or a fieldname of a data field containing an integer value. The <units> may be one of the following abbreviations:

```
NM -- Nautical miles  
MI -- Statute miles  
KM -- Kilometer  
M -- Meters
```

All of the <symbol options> discussed earlier are applicable to either the symbol or CIRCLE definition. Any optional parameter not specified in the

GIPSY WORLD2 MAP FILE
EXAMPLE OF GEOGRAPHIC CIRCLES

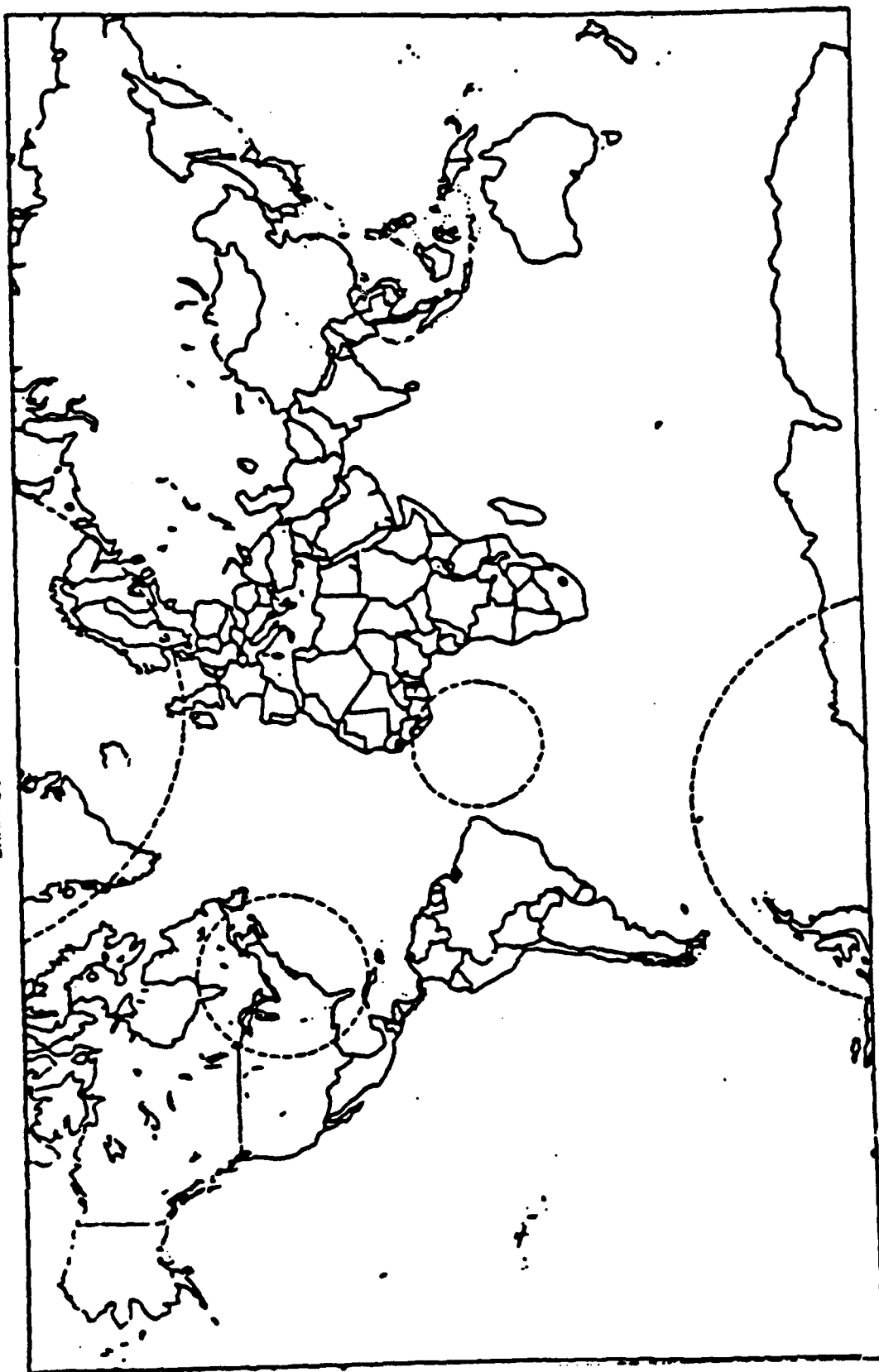


Figure 3-39. Geographic Circles

definition will default to the value assigned on the SYMBOL TABLE statement if they were specified there. Otherwise, the current character size will be used, no protection will be performed, solid lines will be used and the symbol definition name will be blank (i.e., " ").

The IF clause allows a condition to be defined and to describe when that symbol definition will be acted upon. The conditional expression has the same syntax and semantics as described in the retrieval section except that this conditional expression must be separated from any information following it by a semicolon. The conditional expression dictates that the symbol is to be plotted only when the condition is met. If no condition is specified, every record in the QDF is qualified for that symbol. There is no limit to the complexity of individual conditional expressions. There is no limit to the number of symbol definition statements which can be specified in the symbol table.

As an example, let's define a symbol table which will plot the location of military airfields and Navy ships. We also want to display the name and runway length of each airfield, and the name and hull number of each ship.

Assuming that this information is in our data file and that the FDT has these fields defined, the following statements will produce the desired display:

```
SYMBOL TABLE (COORD=LOCATION, PROTECT, SIZE M).
  "*" NAME LOC, MARKER, CENTER, SIZE L.
  NAME.
  LENGTH IF TYPE = AFD.
  HULL-NUMBER IF TYPE=SHIP.
END.
MAP FILE WORLD2, WINDOW CONTAINS FLORIDA, C-BH.
MAP WITH COAST, BOUNDARIES, STATES (LINE DOT-DASH).
SET PROTECT MAP ON.
```

The result is illustrated in figure 3-40.

3.4.2.1.1 User-Defined Graphic Characters. GIPSY provides a capability to define graphic characters which will replace and/or augment the GIPSY graphic characters ("A" thru "H") that were discussed in section 3.4.2.1. Graphic characters are described to GIPSY with a series of moves and draws and accessed utilizing the "<character>" notation. Up to 61 graphic characters can be defined to GIPSY at one time utilizing numeric, alphanumeric and special characters. Graphic characters are described in a 12 by 15 grid, where the x-axis is 00-12 and the y-axis 00-15. The point of origin, 00,00, is the lower left corner of the grid. A move to 0000 is assumed at the start of each character definition. The DEFINE character syntax is as follows:

DEFINE	CHARACTER	"<single character>"	MOVE	<xyxy>	...
	CHAR		M		
			DRAW		
			D		

UNCLASSIFIED

SELECTED AIRFIELDS AND SHIPS

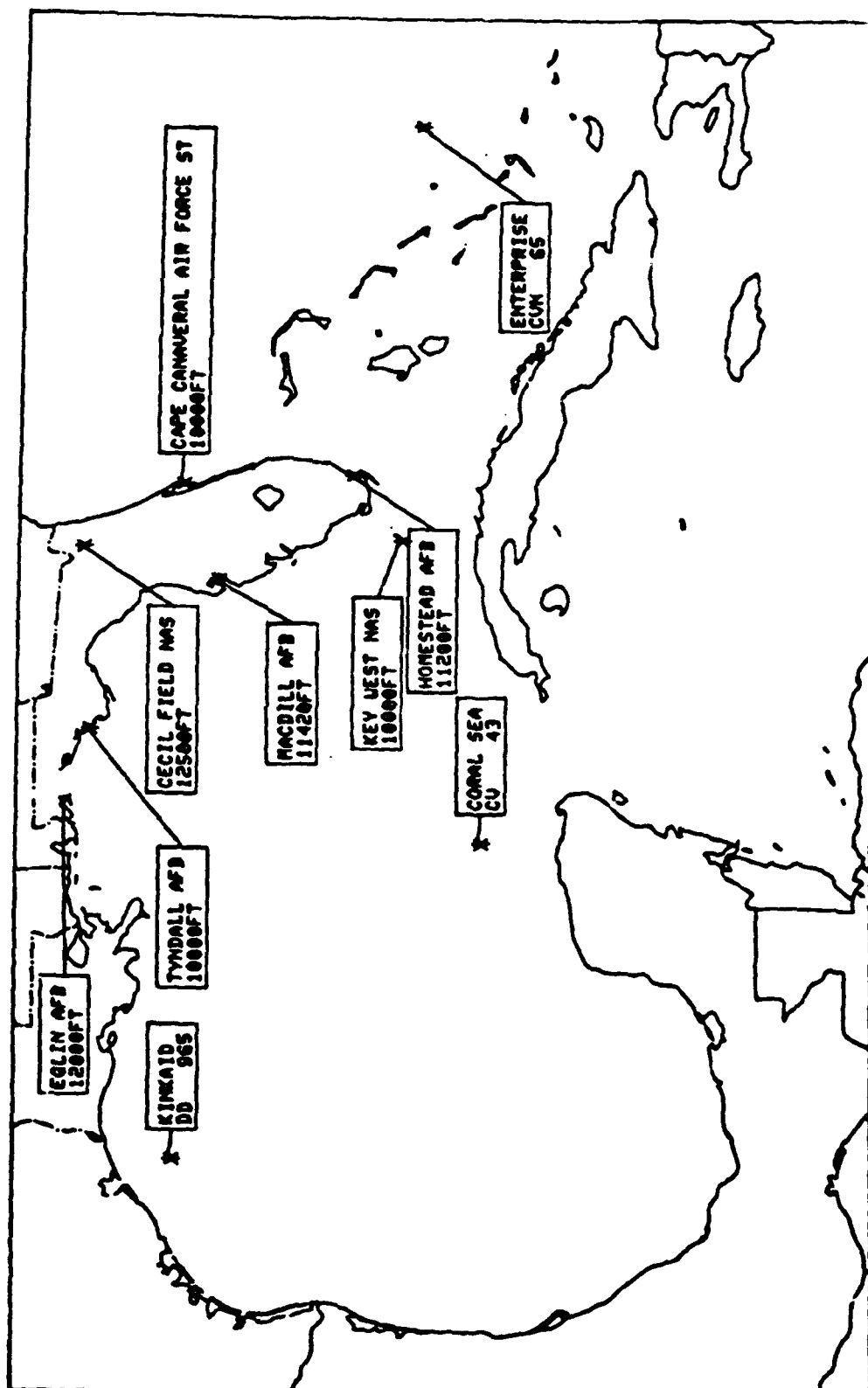


Figure 3-40. Standard System Plot

```

[ MOVE <xyy>
  M
  DRAW
  D

```

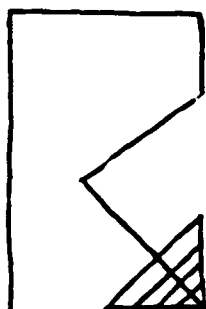
```

[ END
  E

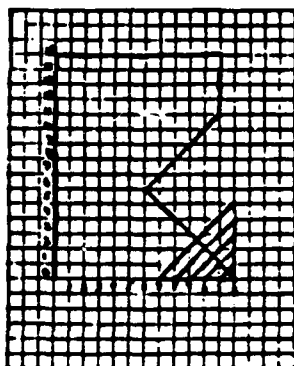
```

The <single character> can be numeric, alphanumeric or a special character.
 The <xx> value range is between 00-12 and the <yy> range between 00-15.
 Therefore if you wanted to draw to x location of 11 and y location of 15 you
 would specify either DRAW 1115 or D 1115.

The following illustrates how a single character is developed from sketching
 stage to actual GIPSY output.



rough sketch



sketch on graph paper

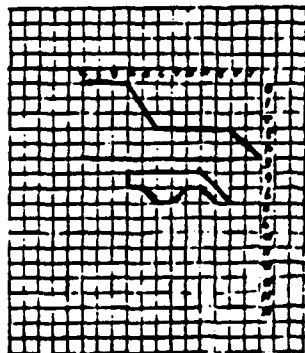
```

1000 DEFINE CHARACTER "\T"
1010 D 0015, D 1115, D 1111, D 0606,
1020 D 1200, D 1205, D 0700, D 1200,
1030 M 0800, D 1204,
1040 M 0900, D 1203,
1050 M 1000, D 1202,
1060 M 1100, D 1201.

```

GIPSY commands

An additional feature of this capability is the ability to string character definitions together. This is especially useful when you need a wide symbol. The following example illustrates how characters are defined individually but "strung together" at display time.



rough sketch
dotted line indicates how
this character will be
divided into 3 parts and
described as three
individual characters

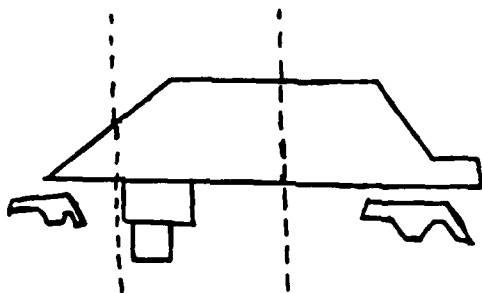


Diagram - Part 1

```
1000 DEFINE CHAR "\U"
1010 M 1200, D 0900, D 0703, D 0203,
1020 D 0005, D 1205, M 0406, D 0208,
1030 D 0308, D 0407, D 0507, D 0608,
1040 D 0708, D 0807, D 0907, D 0906,
1050 D 0406.
```

GIPSY commands
for Part 1

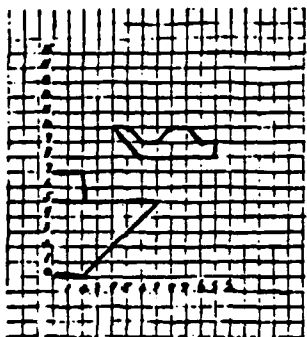


Diagram - Part 2

```

1060 DEFINE CHAR "\V"
1070 D 1200, M 0005, D 1205, M 0605,
1080 D 0607, D 1207, D 1209, D 0809,
1090 D 0807, M 0909, D 0910, D 1011,
1100 D 1109.

```

GIPSY commands
for Part 2

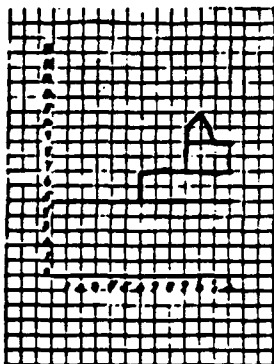


Diagram - Part 3

```

1110 DEFINE CHAR "\W"
1120 D 0200, D 0705, D 0005, M 0205,
1230 D 0207, D 0007, M 0608, D 0410,
1240 D 0510, D 0609, D 0709, D 0810,
1250 D 0910, D 1009, D 1109, D 1108,
1260 D 0608.

```

GIPSY commands
for Part 3

DISPLAY SYMBOL "\U\U\W" SIZE JUMBO.



Actual Display command
and end result.

The last display command demonstrates what is meant by "stringing" characters together. The helicopter carrier was displayed by referencing "\" characters U, V, and W.

GIPSY's default special characters stored in "\" characters A-li can be overwritten simply by defining new characters and restoring those characters in the predefined locations.

3.4.2.2 Building a Track Plot. A track in GIPSY is defined as a series of geographic coordinates connected in succession by a straight line or Great Circle segments. If we were to draw lines connecting each reported position of the aircraft AIR FORCE 1 to the next reported position, the result would be a track of AIR FORCE 1. The movement of any type of unit, ship, ground unit, or aircraft is very effectively shown as a track.

The syntax of the track table is very similar to that of the symbol table; however, there is a significant difference in the semantics. Like the symbol table the track table is a block structure bounded by the declarative TRACK TABLE and the terminator END.

Specifically,

```
TRACK TABLE [CONTINUED] [( <track options> )].  
  
    <track identifier> [<track options>] [IF <condition expression>] .  
    .  
    .  
    .  
END.
```

The CONTINUED option allows a previously specified track table to be reopened and new definitions appended to the end of the table.

The <track options> are similar to the <symbol options> on the symbol table. In fact in both syntax and semantics the <symbol options> are a proper subset of <track options>. The options are:

LINE <line type>	LOCATION
WIDTH <line width>	SIZE <size option>
DIRECTED ON POINT	COLOR <color name>
TRACK	PROTECT
GREAT CIRCLE	GROUP SEPARATELY
DISTANCE	NOT GROUPED
CUMULATIVE DISTANCE	NAME <track name>
TOTAL DISTANCE	MARKER
AZIMUTH	COORDINATE = <fieldname>

The words used to activate the options, for the most part, describe the functions to be performed. They are:

LINE <line type> describes the type of line to be used for relocation lines, the box around a protected list, and a geographic track. This option uses <line type> as previously defined in section 3.4.1.1.

DIRECTED specifies that an arrowhead is to be drawn as a direction of movement indicator; DIRECTED ON POINT causes the direction indicator to be displayed at each point; DIRECTED ON TRACK will cause the indicator to be displayed at the end of the completed track.

GREAT CIRCLE specifies that the points on the track are to be connected by great circle paths.

DISTANCE specifies that the distance between each point is to be displayed at the point.

CUMULATIVE DISTANCE specifies that the cumulative distance from the start of the track is to be displayed at each point.

TOTAL DISTANCE specifies that the total distance from the start of the track is to be displayed at the end of each track.

AZIMUTH specifies that the azimuth from one point to the next is to be displayed (at the start point of each segment of the track).

LOCATION specifies that the coordinate location of each point be displayed.

SIZE <size option> specifies the character size to be used for textual output associated with the track. Character sizes for <size options> defined previously apply.

COLOR <color name> specifies the color to be used in drawing the track and all information associated with the track. The <color name> parameter is the same as discussed on the MAP Statement in section 3.4.1.1.

PROTECT requests that all text associated with the track (e.g., azimuth, distance, etc.) be protected from overprint as described in the symbol protection discussion (section 3.4.2.1). Additionally, the track itself will be protected from overprint if possible. It will not be relocated.

GROUP SEPARATELY requests all protected collocated textual information belonging to different track definitions to be grouped in separate lists for protection purposes. The default is to protect all collocated information in a single list regardless of its definition.

NOT GROUPED prevents the grouping of all collocated textual information for protection purposes.

NAME <track name> specifies that the text string following the key word NAME is to be used as a label to refer to the defined track at some later time. If not specified the <track name> will default to track identifier .

MARKER specifies that the track labels be written at the track point and not be protected.

COORDINATE <fieldname> identifies the field which contains the coordinate data to be used with the current track definition. If this parameter appears on a TRACK TABLE statement, the specified field becomes the default coordinate when one is not specified on the individual definitions.

The individual track options should be separated by commas. Any number of the options may be specified. There is no required order. If conflicting or ambiguous options are specified, the last one specified will be used if possible.

On the individual track definitions, the <track identifier> can be either a literal value enclosed in quotes or a field name. Each track definition statement in the track table defines one or more tracks. If a literal value is used as an identifier, one track will be generated. If the track identifier is a field name, then the content of that field is the actual track identifier and all data records occurring in sequence which have identical contents in the track identifier field comprise a single track. When the field contents change, a new track is established. A conditional expression may be included in the track definition to qualify data for the track. If a conditional expression is attached to a track, the conditions must be satisfied before the record will be considered for inclusion within the track. If any track option occurs after the conditional expression, the conditional expression must be terminated by a semicolon.

Note that if a literal is used as a track identifier with a conditional expression, data records from anywhere in the file will be pulled together as a single track. On the opposite extreme, when we use a field name as a track identifier with no condition attached, a new track will be generated each time the contents of the track identifier field changes.

The sequence in which the points are connected depends upon the order of the data records comprising the tracks and the conditions attached to the definition.

The following example shows a track table which will take a series of ship positions and produce a plot showing the ship's movement. The result is shown in figure 3-41.

```
TRACK TABLE (COORD=LOCATION, PROTECT, SIZE L).  
  NAME DIRECT ON POINT, DISTANCE, LINE DASHED; IF TYPE=SHIP.  
END.
```

Whenever the contents of field NAME changes (and the conditional expression is met) a new track is started. Each subsequent QDF record that has the same ship name in field NAME will add another leg to the track. The SORT statement (section 4.2) can be used to produce a QDF with records in the order required to produce the tracks (e.g., SORT ON NAME, DATE.).

UNCLASSIFIED
SELECTED SHIP TRACKS

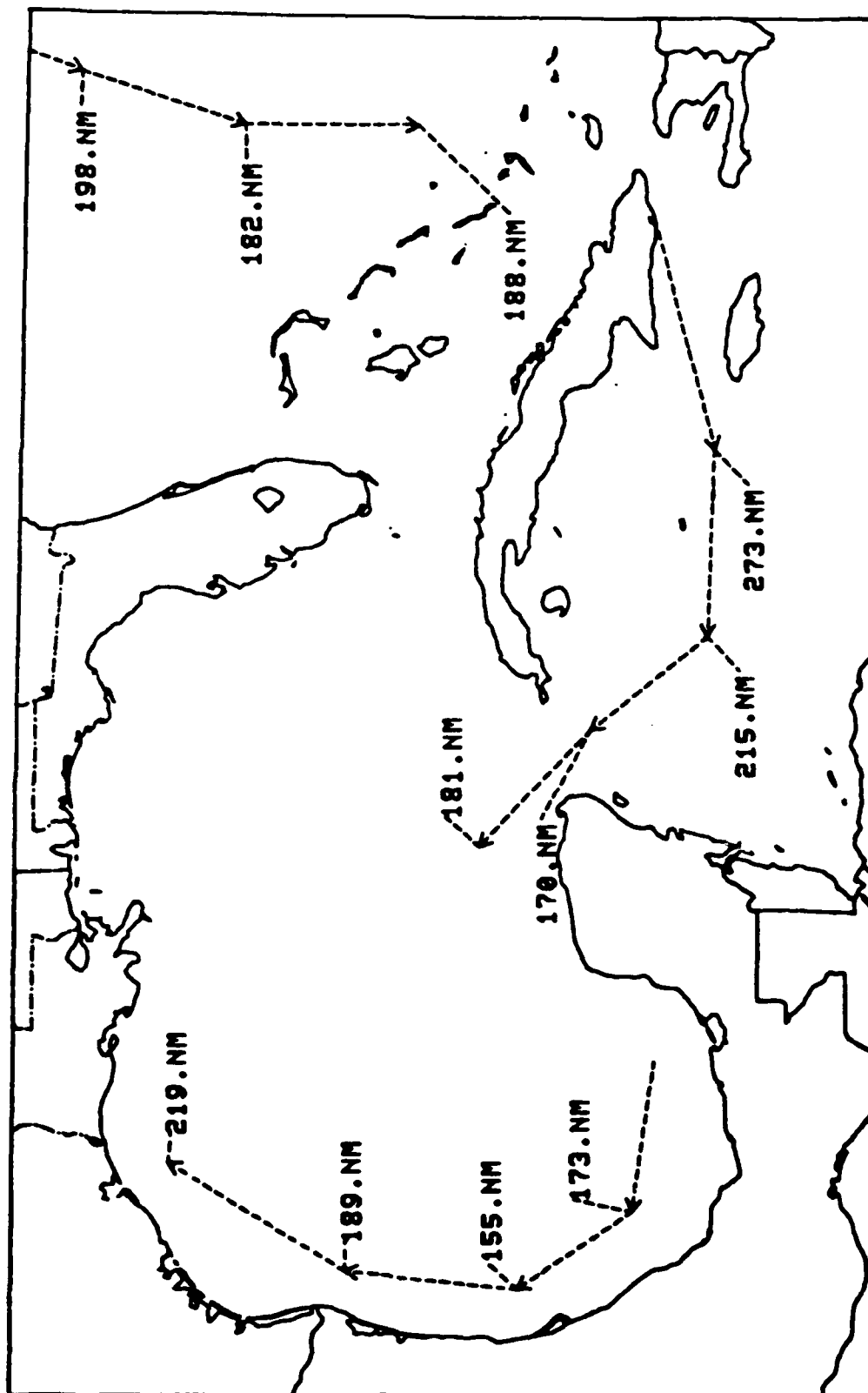


Figure 3-41. Track With Distance Option

3.4.2.3 Synthesizing the Geographic Data. Once all the specifications for building the geographic displays are specified, the data selection and synthesis are initiated by the command RUN. This will cause the user data file to be read and the data structures for the geographic display to be built.

When complete, the user is automatically placed in the geographic mode (GEOMOD) to accept commands which will DISPLAY the map.

Let us take the symbol plot and track plot of figures 3-37 and 3-38 and produce a report containing both symbols and tracks. The complete set of GIPSY commands needed to accomplish this is shown below, with the result illustrated in figure 3-42:

```
FILE 837IDPX0/GIPDEV/ALAN/UM-DATA
FDT 837IDPX0/GIPDEV/ALAN/UM-FDT
SYMBOL TABLE(COORD=LOCATION, PROTECT, SIZE M).
    "*" NAME LOC, MARKER, CENTER, SIZE L; IF NAME COMPLETE.
    NAME IF TYPE-AFD.
    LENGTH IF TYPE-AFD.
    HULL-NUMBER IF TYPE = SHIP AND NAME COMPLETE.
    DATE IF DATE NE " ".
END.
RUN

MAP FILE WORLD2; WINDOW CONTAINS FLORIDA, C-BH;
    WITH NORMAL, STATES(LINE DOT-DASH).
SET PROTECT MAP ON.
SET PROTECT TRACKS ON.
TITLE (SIZE L) "SELECTED AIRFIELDS AND SHIPS";
    (SIZE L) "(SYMBOL AND TRACK PLOT)".
CLASS UZZ.

DISPLAY MAP.
```

3.4.3 Maps and Data Displays. The interactive geographic display mode is entered by one of three ways: (1) by specifying GIPSYG as the GIPSY execution command, (2) by building a symbol table or a track table (as discussed in section 3.4.2), and then entering a RUN command or, (3) by transferring to the geographic display mode from some other mode via the TRANSFER command.

When in this mode, any GIPSY statement other than those associated with data file allocation (section 3.7.4), file description (section 3.2.5), data alteration (section 3.2.6 through 3.2.11) and tabular reports and graphs (section 3.3) may be entered. Note that symbol/track tables may be specified in this mode. However, only fields previously referenced are available.

This section will discuss the syntax and semantics of displaying a geographic display built from a user data base, of building and displaying a new display from keyboard inputs or adding to a data base display, and manipulating the current geographic display.

UNCLASSIFIED

SELECTED AIRFIELDS AND SHIPS
(SYMBOL AND TRACK PLOT)

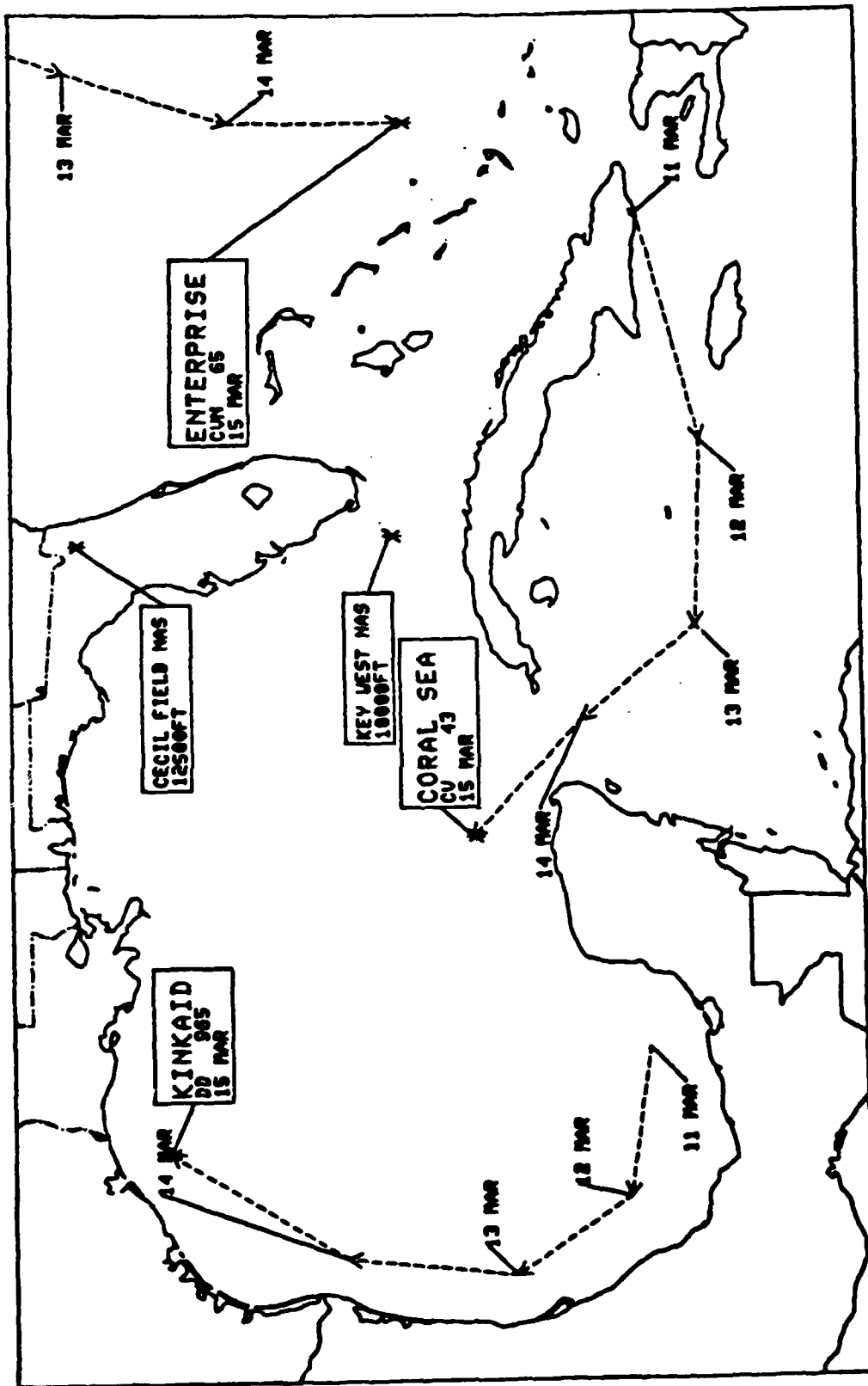


Figure 3-42. Protected Symbol and Track Plot on WORLD2 Map File

DISPLAY [MAP [<map statement options>]].

If the command issued was "DISPLAY." (i.e., MAP options not included) the current display is refreshed on the screen with all temporary and permanent modifications integrated into the display. Protected information will be readjusted if required. When MAP is added to the command (i.e., DISPLAY MAP) the display is refreshed with all temporary modifications deleted from the display. A temporary modification is any modification to the display definition or attributes produced by a DISPLAY or a ZOOM command.

The <map statement options> may be added to the above statement to produce a temporary modification to the map parameters (as specified or defaulted) on the MAP statement. Any options appearing on the MAP statement definition (section 3.4.1.1) may be used on the DISPLAY MAP statement.

3.4.3.2 Interactive Display Building and Modification. The user can use the keyboard and graphic cursors to add any information to the display which can be specified in a symbol or track table. These modifications to the display may be temporary or permanent. Temporary modifications can be redefined to be permanent.

Track, symbols, geographic circles, coordinates of specified locations, azimuth, and computed distances can be added to the display using the general form

```
{
  DISPLAY
  GENERATE }    <display details and specific options> .
```

Information added with DISPLAY is eliminated when a DISPLAY MAP is issued. Information added or referenced with a GENERATE is kept. Display details and specific options are discussed in detail below.

3.4.3.2.1 Adding Tracks. Tracks may be added to the display using the following syntax:

```

{ DISPLAY }
{ GENERATE }

{ TRACK FROM { <location> }
              +
              { <location> }
              +
              <track options>
              NAMED <list of track entry names> }

```

The <location> may be a coordinate or a name from the location table, the

window table or Appendix H or I. The phrases FROM <location> TO <location> define the end points of a track to be added to the geographic display. If the TO phrase is omitted or <location> is replaced by a + (for cursor) the graphic cursors will be activated to allow "TO" locations to be specified via the graphic cursor. The cursor must be positioned to the desired location and a character must be entered to signify to GIPSY to read the cursor location. The track will be plotted immediately from the "FROM" location TO the graphic cursor. The "*" (and shift "*") has been designated as a continuation character. Hence GIPSY will reactivate the cursor for more "TO" values until some other character is entered. The last "TO" value will be treated as the new "FROM" value. This will allow the user to specify a track with as many points as he desires. If FROM location is not specified or location is replaced by a "+", the first cursor location specified by the user will be treated as the "FROM" value. It follows then that if neither FROM nor TO is specified the user is expected to use the graphic cursor to identify all the points on the track.

The track options recognized here are the same as those described in the track table discussion in section 3.4.2.2.

The track on figure 3-43 was produced using the command:

```
DISPLAY TRACK DIRECTED ON POINT, CUMULATIVE DISTANCE, PROTECT, SIZE M.
```

The track was then displayed using the activated graphics cursor.

The requested overprint protection may leave something to be desired because we added tracks to a completed display which can conflict with information already protected on the display. By using a simple:

```
DISPLAY.
```

GIPSY will redisplay the current display and redo the print protection to accommodate the added details. If we issue a DISPLAY MAP command now those tracks will be eliminated...lost forever.

The NAMED clause on the DISPLAY or GENERATE statement is used to recall previously created tracks. All tracks identified in the list of <track entry names> but not on the display (because it was previously excluded) will be added to the display. When DISPLAY is used identified tracks are temporarily added to the display; when GENERATE is used they are permanently added to the display. If a GENERATE statement is used with a temporary track or set of tracks, the track or set of tracks will be made permanent.

The temporary track named TRKD1 created in the foregoing illustration may be preserved by issuing the statement:

```
GENERATE TRACK NAMED TRKD1.
```

UNCLASSIFIED

SELECTED AIRFIELDS AND SHIPS (ADDED TRACKS, SYMBOLS, AND CIRCLES)

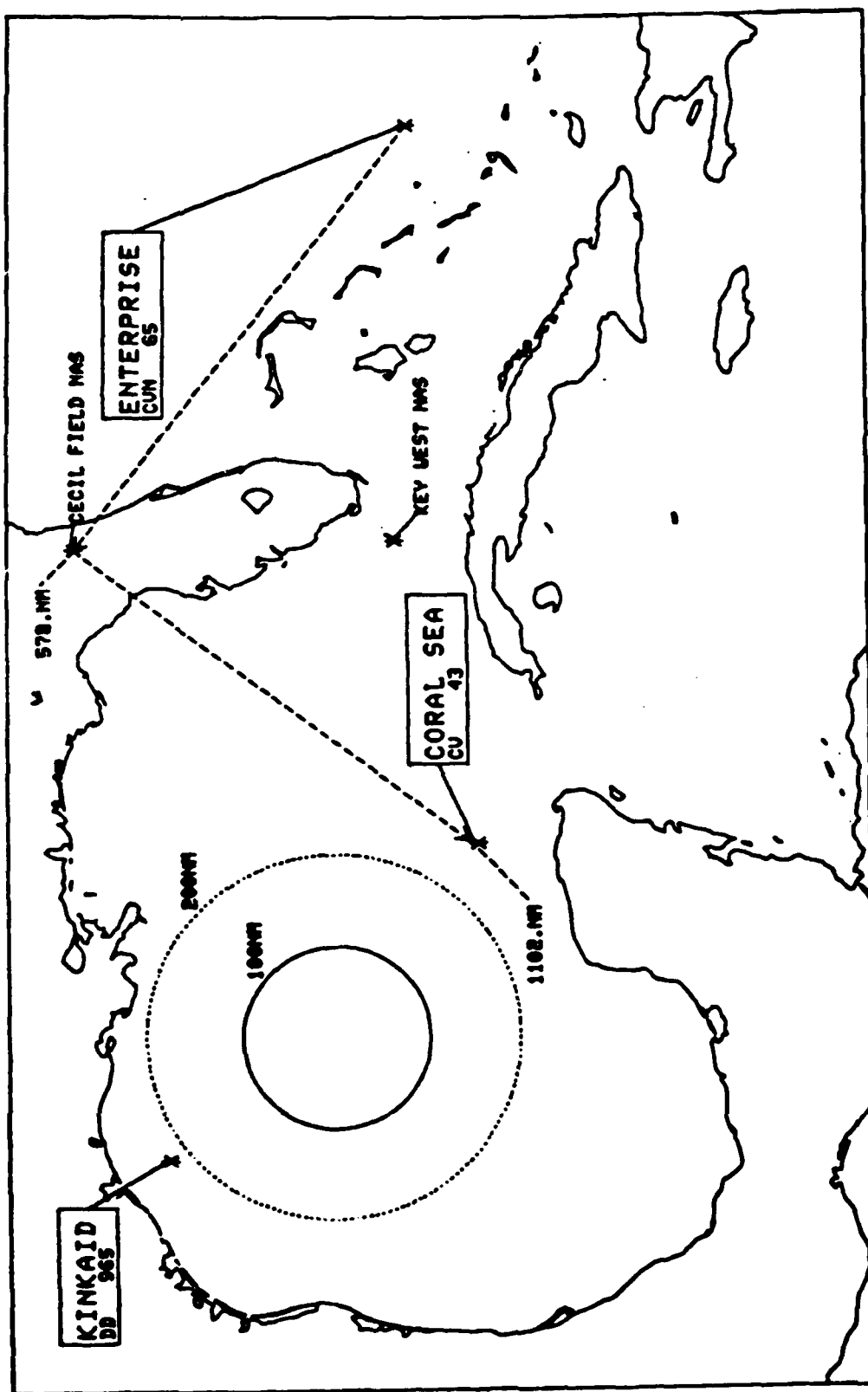


Figure 3-43. Keyboard Supplied Tracks, Circle, and Symbol

A cleaned up display is produced by:

DISPLAY MAP.

Since we didn't save the Great Circle track from San Francisco to Georgia it was lost. It could have been saved by generating the track named " " (all blanks) prior to issuing the DISPLAY MAP command.

3.4.3.2.2 Adding Symbols. Symbols may be added to the display with the following syntax:

```
{ DISPLAY } SYMBOL { "<symbol>" [AT <location>] [<symbol options>] }  
{ GENERATE }      { NAMED          <list of defined symbols> }
```

The character string following the word SYMBOL specified by <symbol> will be plotted on the display at the specified location. The item <symbol> may be an alphanumeric string up to 132 characters in length. However, the literal must all appear on a single input line. The string of characters may contain any of the special GIPSY graphic characters (A, B, C, etc.).

The AT phrase defines the location at which <symbol> will be plotted. The location may be a coordinate or name from the location table, window table, Appendix H or Appendix I. If an area name is used, the geographic center of the area will be used as the location. If <location> is not specified or is replaced by "+", the graphic cursor will be activated for the location of the symbol. The cursor must be positioned to the desired location and a character must be entered to notify GIPSY to read the cursor location. The "*" is used as a continuation to specify multiple locations as was described for adding tracks. The circle labels on figure 3-43 were produced by:

```
DISPLAY SYMBOL " 100NM".
```

```
DISPLAY SYMBOL " 200NM".
```

The <symbol options> recognized here are the same as those described in the SYMBOL TABLE discussion in section 3.2.2.2.

The NAMED clause on the DISPLAY symbol or GENERATE symbol statement is used to recall previously created sets of symbols. All sets of symbols identified in the list of symbol entry names (but not currently displayed because it was previously excluded) will be added to the display. When DISPLAY is used the symbols are temporarily added to the display; when GENERATED is used they are permanently added to the display. If GENERATE is used with temporary symbols (i.e., previously defined on a DISPLAY STATEMENT) those symbols will be made permanent.

3.4.3.2.3 Adding Circles. Geographic circles may be added to the display with the syntax:

$\left\{ \begin{array}{l} \text{DISPLAY} \\ \text{GENERATE} \end{array} \right\}$ CIRCLE $\left[\begin{array}{l} [\text{OF}][\text{RADIUS}]<\text{radius}<\text{units}> [\text{AT } <\text{location}>][<\text{symbol}>] \\ \text{options}> \end{array} \right]$ NAMED <list of defined circle>

DISPLAY and GENERATE causes the created circle to be temporary or permanent as described in adding tracks and symbols. In fact, the DISPLAY or GENERATE circle is treated as a symbol with the additional requirement for radius of the circle. The <radius> must be specified as an integer value. The units must be:

NM for nautical miles,
 MI for statute miles,
 KM for kilometers, or
 M for meters.

The <location> following AT designates the geographic center of the circle. The location may be a coordinate or a name from the location table, window table, appendix H or appendix I. If a <window name> is used the geographic center of the area will be used as the location. If this clause is omitted the graphic cursor will be activated for the center of the circle. The user then repositions the cursor to the desired location and specifies the location by transmitting a character. If the transmitted character was an asterisk (or shift asterisk) the circle will be drawn and the cursor will be reactivated for another location. When any other character is transmitted GIPSY proceeds to the next command without reactivating the cursor. If the RADIUS option is also omitted, the graphic cursor will be activated twice. The first time for the center of the circle, the second time for a point on the circle's circumference.

The <symbol options> are the same as those described in the SYMBOL TABLE in section 3.4.2.1.

The circles on figure 3-43 were produced by issuing the following commands:

DISPLAY CIRCLE RADIUS 100NM.
 DISPLAY CIRCLE RADIUS 200NM, LINE DOTTED.

3.4.3.2.4 Adding Geodetic Information. The coordinate value for locations, the azimuth from one point to another, and distances from a point to one or more other points can be added to an existing display. The syntax for DISPLAY and GENERATE commands for these Geodetic computations and display is

$\left\{ \begin{array}{l} \text{DISPLAY} \\ \text{GENERATE} \end{array} \right\}$
 $\left\{ \begin{array}{l} \text{LOCATION} \\ \text{AZIMUTH} \\ \text{DISTANCE} \\ \text{CUMULATIVE DISTANCE} \\ \text{TOTAL DISTANCE} \end{array} \right\}$
 $\left[\begin{array}{l} \text{SIZE } <\text{size option}> \\ \text{COLOR } <\text{color names}> \\ \text{LINE } <\text{line type}> \\ \text{PROTECT} \\ \text{GROUPED} \\ \text{GROUPED SEPARATELY} \\ \text{NAME } <\text{name to be assigned}> \end{array} \right] \begin{array}{l} 5 \\ 0 \end{array}$

The parameters specified here are defined in the discussion of track options (section 3.4.2.2).

In fact, this command is processed as a track creation in which the track is not drawn. Note that the geodetic computations are the same as those in the track options.

All functions which affect or control the display of track information affect the geodetic information as belonging to an invisible track.

3.4.3.2.5 Changing the Viewed Area. The area of the world shown on the display may be altered by changing the map window or by zooming in or out from the current displayed area. The viewed area may be permanently changed by resetting MAP WINDOW on a MAP statement. The viewing area may be temporarily changed by displaying the MAP with a DISPLAY MAP statement containing a WINDOW clause (see section 3.4.3.1) or it may be changed by zooming in or out from a current display. The latter is accomplished with the ZOOM command.

The ZOOM command allows you to pick any area of the display and zoom in on it for a more detailed look at the subject area or allows you to zoom out from a designated area to pull in a larger area at the price of less distinction among the details.

Consequently, additional map detail will be added to the display as a smaller geographic area fills up the display; detail will be deleted as a larger geographic area fills up the display. Overprint protection will be automatically recalculated to maintain the objective of the protection.

The syntax of the command is:

```
ZOOM  [ IN  [ <lower left coordinate>, <upper right coordinate>
      [ OUT [ <coordinate of new center> [BY] <magnification factor>
      AT   [ <magnification factor>
      BY ] ] ]
```

There are several ways of performing a zoom operation: by using the cursor to define the area to be zoomed in on; by specifying the coordinates of a rectangle defining the area to be zoomed in on; by specifying a new desired center point and a magnification factor, etc. The words IN, OUT and BY can be added to the command for readability but have no effect on the type of zoom performed.

If the statement contains no optional clauses GIPSY will activate the cursor twice, once for the lower left and once for the upper right corner. The coordinates (on the AT phrase) may be used to specify the opposite corners of the rectangle. (On the Textronix 4014-1 the graphic cursor is a very thin line running the length of the screen in both vertical and horizontal directions. Thumb wheels are used to position the graphic cursor and the space bar is used to transmit the location.) Since the horizontal and vertical lines of the graphic cursor can define two sides of the desired

rectangle, two cursor positions are required to define the four sides of the area. For example, if the statement

ZOOM.

is entered, the graphic cursor will appear on the screen. You must use the cursor to the desired location and hit the space bar. GIPSY acknowledges by drawing dashed lines across and down the screen at the entered cursor location; then the cursor reappears to allow the opposite corner of the rectangle to be specified in a similar manner. After this is done, the area within the dashed rectangle is adjusted and rescaled to fill up the screen. (Additional map detail will be included in the display to insure a full display for rectangles which do not fit the screen as specified.) The screen is cleared (if AUTO COPY is on, a copy will be made first) and the described area and associated data is displayed.

Alternately, coordinates of the lower left and upper right corners of the desired area could have been specified. In this case no cursors or dashed lines will appear. The area will be immediately rescaled to fit the specified rectangle. For example:

ZOOM IN AT 442315N1253320W, 480000N0950000W.

The second method of accomplishing a zoom requires a specification of the new center of the display and a magnification factor. This option is very useful in recentering the display over a particular area of interest. The new center of the display may be specified either with the graphic cursor or from the keyboard by specifying the actual coordinates of the display. For example:

ZOOM AT 2430N12600W BY 1.5.

If the BY clause is omitted GIPSY will prompt you for it. The magnification factor may be either integer or floating point; it may be negative or unsigned (positive). A magnification factor of 1.0 causes the display to be reproduced at the same scale but with a new center; between 0 and 1 produces a proportionately smaller scale (i.e., larger area--zoom out); greater than 1 produces a proportionately larger scale (zoom in). Negative numbers cause a zoom out just as fractional numbers (between 0 and 1) do, only these proportions may be easier to follow.

The cursor can be used to specify the center point in two ways. First, specifically ask for it by specifying a "+" as the coordinate of the new center of the display (e.g., ZOOM AT + BY -3.0).

The graphic cursor will be automatically activated to allow you to specify the center of the zoom area. Similar results can be achieved by omitting the AT clause in entirety:

ZOOM BY 0.333.

A ZOOM may increase the amount of user data displayed beyond that presented in the original display (i.e., a ZOOM can be used to expand the map window on user data in conjunction with an expansion on the map area). However, the ZOOM does not add data that was not referenced or qualified.

Be sure to include a period to end the sentence or GIPSY will respond with a prompt character to request more input.

3.4.3.2.6 Limiting the Data Displayed. When all the data that was used to produce a display is on the screen at once, clutter can be a problem. The clutter can be reduced by restricting the number of QDF records displayed (see section 3.4.4), or by limiting the amount of information displayed for each record. A LIMIT statement is available to allow the user to specify exactly which information subsets are to be included in the display. A LIMIT statement has been provided to allow the specification of exactly which subset of data is to be included in the display. The LIMIT statement operates on the three basic sets of geographic information--symbols, tracks and circles. The syntax for the statement is:

```
LIMIT  {SYMBOLS} [TO] [NONE  
        {TRACKS}  ALL  
        {CIRCLES} *  
                <subsets of data names>]
```

The list of subset of data names referenced above may be a single data name and/or a range of names in the form:

<first subset data name desired> THRU <last subset data name desired>

SYMBOLS, TRACKS, or CIRCLES define the respective set of information to be limited. NONE indicates that no information from the specified set will be displayed. The statement

LIMIT SYMBOLS TO NONE.

will preclude any symbol from appearing on the display until another LIMIT statement overrides this request:

LIMIT SYMBOLS TO ALL.

will allow all symbols to be displayed. The "*" means the same as "TO ALL".

The <names of subsets of data> is a list of names assigned in the symbol or track table definitions or assigned on a DISPLAY or GENERATE statement. Each name in the list should be separated by a comma. The first subset data name desired and the last subset data name desired are defined as the name implies. The THRU operator causes all names in between (inclusive) to be included in the limit.

A subset that is not displayed because it has been limited out may be forcibly

displayed by specifically requesting it on a DISPLAY or GENERATE statement. A specific display request temporarily adds the identified subset to the limit definition.

3.4.3.2.7 Listing the Display Data. It is often desirable to obtain a printed listing of the retrieved data which is contained in the QDF. This is normally accomplished with the QUALIFY and PRINT commands (section 4.1.2).

The LIST command, however, allows the user to graphically select which records are to be listed. The syntax is:

```
LIST  [ (SIZE  <size option>)      ] 4
      [ NAME "<name>"                ] .
      [ FIELDS <fieldname>, ...     ]
      [ WITHIN CIRCLE <circle name> ] 0
```

Two sets of graphic cursors will be made available to bracket the area from which all data items will be listed. The graphic cursors are used identically to the ZOOM when specifying a rectangle around the area of interest.

The fields of all data records associated with points within the defined rectangle will be displayed on the terminal. If the FIELDS option is used, only the contents of the specified field names will be listed; otherwise, all QDT fields will be displayed.

If the WITHIN CIRCLE clause is used, GIPSY will list those records which fall within a previously defined and named circle. No graphic cursor will appear.

The NAME clause will append the specified <name> to the top of each listing when used in conjunction with the EXTRACT command.

The EXTRACT command allows the output of the LIST command to be written to a sequential file. The syntax is:

(1) EXTRACT [LIST] ON <cat/file string>.

(2) EXTRACT { ON
 OFF } .

Once the EXTRACT output file has been specified, all LIST commands will automatically be written to that file as well as to the screen. Each new listing will be appended to the previous listing. The NAME clause in the LIST command can be used to give each listing a separate header. The command "EXTRACT OFF." will suspend writing to the file, whereas "EXTRACT ON" will cause it to resume.

3.4.3.2.8 Adding Textual Information. Once a geographic report has been produced, it is possible to add blocks/group of textual information. This is accomplished by using the TEXT commands previously discussed in section 3.3.3.3.10. All of the capabilities discussed in that section are available

for use in the geographic module.

3.4.4 Secondary Data Retrieval. The creation of geographic displays for command and control must be an interactive process. This includes being able to dynamically alter the selection of data to build the display. (Of course you can always restart the query and alter the RETRIEVE from the data file.) This capability is necessary if you wish to qualify a subset of the data already retrieved for display. GIPSY provides a QUALIFY statement to select portions of the data from the already retrieved data (i.e., a secondary retrieval). The syntax for the secondary retrieval is:

```
QUALIFY { [IF] <conditional expression> }  
        ALL
```

Only the data which meets the criteria specified in the <conditional expression> will be used in building the geographic display. Unlike the RETRIEVE statement, this statement does not actually eliminate the data. It simply causes GIPSY to ignore any data record which does not meet the QUALIFY conditions. You may alter the effect of a QUALIFY statement simply by issuing another QUALIFY statement.

This capability allows you to retrieve a subset of data to meet a large set of conditions that selectively produce displays by QUALIFYing a subset of the data needed for that particular geographic display.

The keyword IF is optional and is added only for clarity and readability. Recall that the definition of <conditional expression> (section 3.2.6) can consist of logic names from a logic table, logical data fields, etc. These can be used to create readily entered and readily understood QUALIFY statements.

If we had a retrieval which retrieved all the ships in a given area, we could use the LOGIC TABLE defined in section 3.2.7 to perform complex data screening, with a simple and easy to understand QUALIFY statement.

The statement:

```
QUALIFY COMBAT6FLT.
```

will make available only combat ships in the 6th Fleet. This would normally be followed by one or more display commands to display the resulting data. If we wanted everything but combat ships in the 6th Fleet we would enter:

```
QUALIFY NOT COMBAT6FLT.
```

To obtain noncombat ships and all submarines we might enter:

```
QUALIFY NOT COMBAT OR ANAME (1/1) EQ "S".
```

3.5 GIPSY Language Mode Transition

The GIPSY user enters commands or statements in three distinct modes. These modes are commonly referred to as (1) SYNTAX - the retrieval and processing specification mode; (2) GEOMOD - the geographic display building mode; and (3) DISPLA - the tabular report and graph display mode. GIPSY provides a limited capability for transition from one of these modes to another. The command for accomplishing this is the TRANSFER command. Its syntax is:

TRANSFER TO

{
SYNTAX
DISPLAY
GEOMOD
GDR
}

If a transfer is made to the syntax module to rebuild a tabular report GIPSY will reenter the DISPLA mode without destroying the previous report. Any DISPLA action will apply to the previous tabular report which is returned in memory. The newly built tabular report must be loaded via an ACCESS REPORT command which will replace the old tabular report with the new one.

3.6 Picture Processing

This section describes the GIPSY tools available to save and recall complete graphic displays. All the graphic information and attendant alphanumeric information that is displayed on the screen is called a picture. These pictures may be saved on a standard data file for subsequent display. There is some inherent overhead in this process. Consequently picture processing is only activated when specifically requested by the user.

Picture processing is used to create a series of displays which can then be recalled at a future session. Some uses include combining the graphic output from several different GIPSY sessions, or creating graphic displays in the batch environment or on a non-graphic terminal for subsequent display on a graphics terminal.

3.6.1 Saving Pictures. The term "RETAIN" is assigned to the process of saving a picture. This distinguishes the saving of a single picture from the save concept usually associated with a file operation.

Before any picture processing is performed the user must initiate picture processing by issuing the command:

SET PICTURE PROCESSING ON.

This command enables GIPSY's picture processing software which allows each graphic display to be preserved until a subsequent graphic display is produced. Each subsequent graphic display will replace the previous one unless it is retained. When a picture is complete, that picture may be retained and labelled for subsequent recall. The syntax for retaining the current picture is:

RETAIN PICTURE [[NAMED] <picture name>].

If the user chooses to specifically identify each picture, it is accomplished by utilizing the NAMED option and specifying <picture name> as a label of 1 to 12 characters. This label must be enclosed in quotes if it contains blanks or special characters. If the NAMED option is not used the name "PIC n" will be assigned, where n is an integer representing the sequence in which the pictures were saved. A series of pictures may be saved by notifying GIPSY to retain all pictures until further notice. The command to accomplish this is:

SET AUTO RETAIN ON.

This will cause each picture generated to be saved prior to initiating a new picture. Any information added interactively to the display, such as symbols and tracks, will also be saved. The very last picture must be manually retained although the last display will be retained when the DONE command is issued. Automatically retained pictures will be assigned unique names in accordance with the naming convention discussed above. Each picture will be assigned the prefix PIC followed by a number indicating its ordinal position in the sequence of all pictures retained. Duplicate names will not be generated. The automatic retention capability is turned off with the command:

SET AUTO RETAIN OFF.

All pictures will be retained on a system generated temporary file unless a picture file is supplied. If a picture file is supplied it must be a file which already contains GIPSY pictures or it must be an empty random file. If the file already contains GIPSY pictures, new pictures will be retained following the last picture on the file. The command to force GIPSY to use a user supplied permanent file in lieu of the normal system supplied temporary file is:

PICFILE <cfd>

Where <cfd> is the catalog file descriptor of the file to be used for both writing new pictures and recalling old pictures. The standard GIPSY SAVE/RESAVE commands may be used to save the entire set of pictures onto a permanent file. The syntax is:

{ SAVE
RESAVE } PICFILE [ON] <cfd>

3.6.2 Recalling Pictures. A retained picture may be recalled and redisplayed by entering the PLOT command:

PLOT PICTURE [NAMED] <picture name> .

where <picture name> is the name that was assigned to the picture at the time it was retained. The PLOT command can be issued in either the DISPLA or GEOMOD modules.

Picture processing need not be set on in order to PLOT a picture. To plot a picture on a previously saved picture file, the PICFIL <cat/file string> command must be entered first. The interrupt command:

//PICTURE

will list all the pictures available for plotting.

SECTION 4. QUERY PROCEDURES

This section is targeted primarily toward the applications programmer who supports the user in providing the more complex procedures as predefined PCS files. These materials provide a further amplification of the data selection process, field and record modification, sorting and formatted data printing. Several capabilities have been previously discussed, some have not.

4.1 File Query

GIPSY is not a Database Management System (DBMS). While it does powerful data selection and subsetting capabilities it does not provide sophisticated query capability as one would expect of a DBMS. It is primarily targeted toward allowing the user to select the data from his/her database for creation of graphic displays and to review the data that has been or will be selected for inclusion in a graphic or tabular display. A data retrieval, a browse, and data subsetting capabilities comprise GIPSY's file query capability.

4.1.1 Data Retrieval. GIPSY's basic data retrieval is accomplished via the RETRIEVE statement as discussed in section 3.2.8. The RETRIEVE statement can only be specified in the syntax mode. It creates the subset of the data file that is used in all subsequent GIPSY operations. Of course, the user may execute a "TRANSFER TO SYNTAX" statement and respecify the retrieval statement to expand the set of data that is retrieved and saved on the QDF. All data which passed the retrieval conditions are written to the QDF. The QDF may in turn be saved or assigned to permanent file from the start. Saving the QDF and its associated QDT provides a very effective data subsetting capability.

4.1.2 Browsing. GIPSY's browsing capability allows the user to browse through any portion of the data on the user's current QDF (data which passed the RETRIEVE conditions). This capability is accomplished via two statements: QUALIFY and PRINT. QUALIFY sets up a screen to determine which data records will be used. PRINT causes the selected files from the qualified records to be displayed.

QUALIFY and PRINT are actually imbedded within the geographic display capability but does not require that geographic displays be produced. If the GIPSY input sequence does not contain a tabular report block structure, GIPSY transfers control to the geographic module where QUALIFY and PRINT statements may be issued. These statements may be used in conjunction with or independent of the geographic display functions.

The syntax for the QUALIFY statement is:

```
QUALIFY  { [IF]    <conditional expression> } .  
          ALL
```

The IF is optional and may be added for readability. The <conditional expression> is as defined earlier. Recall that the <conditional expression>

may be a set of comparisons or a logic name (from a LOGIC TABLE) or repetitive combinations thereof. This statement sets up a "filter" for all data subsequently processed by the module. The input is the data currently contained on the QDF. QUALIFY does not actually change what is in the QDF, it merely sets up conditions for its processing. Since the data contained in the QDF is only that which has previously passed a retrieval criteria, the QUALIFY represents a secondary retrieval.

You can reference fields on the QUALIFY only if they exist in the current QDF (e.g., they were previously referenced prior to the RUN). The INCLUDE statement may be used to ensure that all desired fields are in the QDT. The QUALIFY statement may be reissued as often as desired. Each subsequent definition replaces the previous one. It filters all data until it is replaced or nulled by the statement "QUALIFY ALL."

PRINT provides the capability to display the contents of selected data items. It provides a semi-formatted listing of the fields identified in the statement. If a QUALIFY has been issued, only the data meeting the QUALIFY conditions are displayed. Otherwise, all data records on the QDF are displayed. QUALIFY does not cause the data file to be read. The statement is simply compiled and set up as a filter. PRINT causes the data to be read and displayed. Each record is displayed as it meets the QUALIFY condition. Consequently, QUALIFY and PRINT produce almost instantaneous results.

The syntax for the PRINT statement is:

```
PRINT [FIELDS <fieldname> ["<header>"] ,...] [<format options>].
```

```
where <format options> are      INDENT  <number of spaces>
                                SPACE    <number of spaces>
```

If no fields are identified (i.e., "PRINT."), all fields in the QDT will be displayed. The report will be formatted with the fieldname centered over the contents of the data field. The report will be automatically formatted for the device from which the command was issued. Pagination will occur first vertically and then horizontally if the fields to be printed exceed the width of the device. Fieldnames may be specified to restrict the data display to the desired fields, to confine the display to the width of the display device, or to force the order of the printed fields. If <header> is included, that information will be used as the header rather than the fieldname. Each fieldname must be separated from the other by a comma. The option INDENT followed by an integer specifies that the entire report will be shifted to the right by that many spaces. The option SPACE followed by an integer specifies the number of spaces between each field list. The default is two spaces.

Since both QUALIFY and PRINT are immediate action commands, QUALIFY and PRINT may be used repetitively to provide a very effective browse capability.

The user's entire database may be subject to GIPSY's browsing capability by treating the data file as the QDF and the FDT as the QDT. That is, issue the

statements:

```
QDF  <user data base>
QDF  <data base FDT>
```

then the entire database is available for instantaneous interactive selection and display via QUALIFY and PRINT.

This browsing capability is only available for a GIPSY-created QDF or sequential user file. The QDF statement cannot point to a nonsequential file.

4.1.3 Subsetting the Database. As has been discussed earlier, the RETRIEVE statement may be used to identify the data records to be included in the subset of the data base. The FIELD TABLE may be used to augment or rearrange the data in the record. The extended record area described by the BUILD FDT may be used to create new fields. SORT (discussed in section 4.2) may be used to rearrange the records. Finally, the RECORD OUTPUT TABLE (discussed in section 4.3.2) may be used to rebuild the organization of the file's data structures. All these functions may be included in composite in the QDF which may be saved as a subset of the user's data base. Its syntax is:

```
{ SAVE }
{ RESAVE } QDF [ON] <cfid>
```

In all cases except for the use of a RECORD OUTPUT TABLE, GIPSY provides a description of the resulting subset as the QDT. The QDT may be saved by issuing the statement

```
{ SAVE }
{ RESAVE } QDT [ON] <cfid>
```

The QDF and QDT may be used as a data file and FDT respectively in a subsequent retrieval. The QDT is stored in FDT format; therefore, it may be used as an FDT. Some applications use this technique to create a proper subset of the master database and then use GIPSY or other software to operate on the subset as it does the master database.

4.1.4 Printing to a File. The output of a PRINT command can also be written to a sequential disk file. The syntax for this feature is:

```
PRINT [<print option>][HEADER "<literal>"][TRAILER "<literal>"] ON <cfid>.
```

The <print option> includes any option from section 4.1.2. The HEADER option allows the user to append a literal to the beginning of the entire report, whereas TRAILER places a literal at the end of the report. Both HEADER and TRAILER are left-justified on the report.

The report can be formatted further by specifying the length and width of each output page, using the command:

SET OUTPUT <width> BY <length>.

The parameter <width> specifies the number of characters in a line of print, and <length> specifies the number of lines on a page. As an example, SET OUTPUT 132 BY 66 could be used to PRINT a report to a file in a format which can later be sent to an online printer. The SET OUTPUT command is canceled by any SET SIZE command.

4.2 Data Sorting

GIPSY provides the capability to sort the data retrieval from the user master file. The syntax for creating a sorted QDF is:

```
SORT ON <fieldname1> [ ASCENDING  
DESCENDING ] [{,} <fieldname2> ASCEND  
DESCEND ... ] .
```

The first field specified becomes the major sort, each subsequent field becomes the next subordinate sort. This capability does not sort the user's database. It sorts only the data comprising the QDF. The fieldnames specified as sort keys may be either fieldnames for data actually in the file or for data provided via the field table. The sort order, ASCENDING or DESCENDING, can be specified for each field. The default is ASCENDING. If either option is specified, it must be separated from the next fieldname with a semicolon.

4.3 Data Modifications

It is unusual to have a functional database whose objective is graphic display. Typically, the database was created and is maintained for some other purpose. The addition of computer graphics and related services offer the opportunity to significantly expand the utility of the data already recorded. This section discusses the tools provided by GIPSY to aid in exploiting existing data. These tools are primarily targeted toward the application programmer rather than the user analyst. The application programmer will use these tools to improve the interface between the user and his database.

It is frequently found that the data structures already in the database do not support the required processing functions or that the data records are not organized to allow effective exploration of the data. To accommodate these anomalies, GIPSY provides commands and block structures to allow the record to be modified at the field level, the file to be modified at the record level, to rearrange the records via a sort functions, and to save the results for a subsequent use of GIPSY or other user applications. These modifications are applied in virtual space and do not actually alter the user's database. The resulting output may be saved and used as an alternate copy of the database, a subset database, a transaction file for a subsequent file update using the applications standard procedures, etc.

These temporary data modifications are particularly useful in correcting discovered database errors "on the fly", and in processing one set of data

based on information retrieval from a different file.

4.3.1 Field Level Data Modification. A block structure called a field table is provided to allow the content of any data field defined in the FDT to be modified. It also allows data to be generated from existing data and inserted in new fields which do not actually exist in the file. The fields modified or loaded by the field table may be used as if they were physically part of the record.

The syntax for the field table block structure is:

```
FIELD TABLE      [ FOR INITIAL
                   WHEN <conditional expression> ] .

<fieldname> = <alphanumeric expression> [IF <conditional expression>].
<fieldname> = (<arithmetic expression>) [IF <conditional expression>].
.
.
.
END
```

The <conditional expression>, <fieldname>, <arithmetic expression> have the same syntax and semantics as defined earlier. Note, however, that the arithmetic expression must be contained within parentheses. The alphanumeric expression is composed of a single fieldname, a literal, or a sequence of fieldnames and literals connected by a colon. All literals must be enclosed in quotes. The colon (:) is the operator which designates a concatenation operation. The concatenation operation combines the two units of information on each side of the colon into a single unit of information. The <fieldname> on the left of the equal sign designates the field to receive the results of the alphanumeric or arithmetic expression on the right hand side of equal sign. However, the operation is performed only when the conditional expression is true. If no conditional expression is specified, the indicated operation is always performed. The field table may contain as many of these field assignment statements as desired. They will be processed in the order specified. Consequently, any statement may use the result of any preceding statement. The same <fieldname> may be the receiving field as often as desired with each independently processed in the order specified. There is no way to unspecify a field assignment statement in the field table once it is syntactically correct.

Every record in the database is processed against each statement in the field table as the conditional expression permits. A conditional expression on the FIELD TABLE statement controls entry into the field table. In other words, if the FIELD TABLE's conditional expression is not satisfied, none of the field assignment statements in that field table will be processed against that record.

A single run may contain multiple field tables. Each field table and

statements contained therein will be processed against each record in the database. The field tables are applied to the database prior to processing the RETRIEVE statement. Since the field table is used to generate the data for the record, use of field names within the field table do not constitute field reference with regard inclusion of that field in the qualified data file (QDF). Fields used in the field table will not be carried forward unless they are otherwise referenced. The field receiving the data determines the attributes of the field table processing. GIPSY will convert the results of the right hand side to match attributes of the receiving field (on the left). The field will be padded with blanks or structured to fit. Note, however, that if a partial field is used as the receiving field, the type will always be treated as alphanumeric and the length will be that of the partial field. Partial fields on the left of the equal sign may be used to replace a portion of the data while leaving the unreferenced portion intact.

The following example illustrates the use of the field table to replace undesirable data in record type A:

```
001 FIELD TABLE WHEN RECORD-TYPE EQ A.
002 DATE="810831" IF DATE EQ BLANK.
003 DATE (5/6)= "30" IF MONTH EQ 04, 06, 09, 11 AND DAY GT 30
004 DATE (5/6)= "31" IF MONTH EQ 01,03,05,07,08,10,12 and DAY GT 31
005 DATE (5/6)= "28" IF MONTH EQ 02 AND DAY GT 28.
006 END.
```

Line numbers were added to the above statements in order to allow them to be referenced in our discussion. (GIPSY will accept statements with line numbers if the statements are being specified in a PCS file.) Line 001 stipulates that this field table will be used only when the record type field of the record being processed contains an "A." If this condition is not satisfied then lines 002 through 005 will be skipped. Line 002 replaces all blank date fields with a valid date. Lines 003 and 004 corrects for erroneous days depending on whether the month is 30 or 31 days. Note the partial field on the left to change only a part of the field in line 003 through 005.

The field table is often used in conjunction with a newly created field. The BUILD FDT block structure may be used to dynamically modify the FDT to add new fields which do not really exist and then the field table is used to load data into those new fields. The interrupt command //FDT shows all the fields already defined:

```
>//FDT
FIELD      POSITION  LENGTH  TYPE  DECPT  QDT  POSITION
RECORD-TYPE 1        1    A
DATE         2        5    A
READINESS    8        5    A
UNITID       20       12    A
EFFECTVNESS  36        8    F      5
>
```

Now the FDT is modified to include a few new fields to be used in a field table:

```
BUILD FDT.  
ADD.  
CLEAR-DATE * A9.  
DAY CLEAR-DATE (1) A2.  
MONTH CLEAR-DATE (4) A3.  
YEAR CLEAR-DATE (8) A2.  
GOOD-COORD * A15.  
END.
```

Note the use of field name (CLEAR-DATE) to provide a starting location for the new fields DAY, MONTH, and YEAR. Note also that an asterisk was used to cause all these "phantom" fields to be appended to the current end of the defined record--wherever it may be.

We can now employ a field table to load data into these otherwise empty fields. The appropriate month is loaded, dependent upon the numeric month code in the field DATE; then the day and year are copied over after appending appropriate blanks (e.g., 830317 would be converted to 17 MAR 83):

```
FIELD TABLE.  
MONTH - "JAN" IF DATE (3/4) EQ 01.  
MONTH - "FEB" IF DATE (3/4) EQ 02.  
MONTH - "MAR" IF DATE (3/4) EQ 03.  
MONTH - "APR" IF DATE (3/4) EQ 04.  
MONTH - "MAY" IF DATE (3/4) EQ 05.  
MONTH - "JUN" IF DATE (3/4) EQ 06.  
MONTH - "JUL" IF DATE (3/4) EQ 07.  
MONTH - "AUG" IF DATE (3/4) EQ 08.  
MONTH - "SEP" IF DATE (3/4) EQ 09.  
MONTH - "OCT" IF DATE (3/4) EQ 10.  
MONTH - "NOV" IF DATE (3/4) EQ 11.  
MONTH - "DEC" IF DATE (3/4) EQ 12.
```

```
CLEAR-DATE (1/3)-DATE (5/6):" ".  
CLEAR-DATE (7/9)-" ":DATE (1/2).
```

END.

The field table may also be used to perform computations and to store the resulting data into fields for subsequent use. While the field table does not have the efficiency and flexibility of a High Order Language (HOL) such as COBOL or FORTRAN, it has the advantage that the function can be imbedded in a GIPSY sequence and used as if it were part of the file without actually changing the file.

The following example is based on a file which (for reasons of the application) has coordinates stored in signed floating point numbers. GIPSY requires a 15 character coordinate. This field table converts from binary floating point to coordinate form:

```

FILE REALDATA
BUILD FDT. ADD.
    LATITUDE 1 F8.4.
    LONGITUDE 9 F9.4.
END.
//
//DEFINE ADDITIONAL FIELDS FOR MAKING CONVERSION
//
BUILD FDT. ADD.
    LATSEC * BI.
    LONSEC * BI.
    DEGLAT * I2.
    MINLAT * I2.
    SECLAT * I2.
    DEGLON * I3.
    MINLON * I2.
    SECLON * I2.
    EW      * A1.
    NS      * A1.
    LATLON * A15.
    GEOCCRD LATLON C15.
END.
//
FIELD TABLE.
//
//CONVERT FLOATING POINT COORDINATE TO SECONDS
//
    LATSEC=(3600*LATITUDE) IF LATITUDE GE 0.
    LATSEC=(-3600*LATITUDE) IF LATITUDE LT 0.
    LONSEC=(3600*LONGITUDE) IF LONGITUDE GE 0.
    LONSEC=(-3600*LONGITUDE) IF LONGITUDE LT 0.
//
//CONVERT SECONDS INTO DEGREES, MINUTES, AND SECONDS
//
    DEGLAT=(LATSEC/3600).
    MINLAT=((LATSEC-(DEGLAT*3600))/60).
    SECLAT=(LATSEC-DEGLAT*3600-MINLAT*60).
    DEGLON=(LONSEC/3600).
    MINLON=((LONSEC-(DEGLON*3600))/60).
    SECLON=(LONSEC-DEGLON*3600-MINLON*60).
    EW="E".
    EW="W" IF LONGITUDE LT 0.
    NS="N".
    NS="S" IF LATITUDE LT 0.
    LATLON=DEGLAT:MINLAT:SECLAT:NS:DEGLON:MINLON:SECLON:EW.

```

```

//
//PUT BACK LEADING ZEROES
//
LATLON(1)="-0" IF LATLON(1)="-" ".
LATLON(3)="-0" IF LATLON(3)="-" ".
LATLON(5)="-0" IF LATLON(5)="-" ".
LATLON(8)="-0" IF LATLON(8)="-" ".
LATLON(9)="-0" IF LATLON(9)="-" ".
LATLON(11)="-0" IF LATLON(11)="-" ".
LATLON(13)="-0" IF LATLON(13)="-" ".
END.

```

The field GEOCORD may now be used as if it were part of the data file.

4.3.2 Record Level Data Modification. Data may be modified at the record level by rebuilding the data structures that comprise the data file. This is accomplished via the GIPSY block structure called the record output table. The record output table allows one to define the fields to be included in a new record and the conditions under which those records are output. The syntax for the record output table is:

```

RECORD OUTPUT TABLE [WHEN <conditional expression>].

FIELDS <fieldname>    <fieldname>    ... IF <conditional expression> .
      <literal name>  <literal name>

.
.
.
END.

```

The optional <conditional expression> on the RECORD OUTPUT TABLE statement controls use of the record output table. If this condition is not satisfied, the record output table will be ignored. If no condition is attached to the record output table, the table is always active.

Each entry in the record output table begins with the word FIELDS and is followed by a series of <fieldnames> and <literals> which define what is to be output to the QDF. The <conditional expression> controls the output of this particular record. If no condition is attached, the described output record is always written.

The presence of a record output table causes GIPSY to suspend the normal process of field inclusion into the QDF. Only those fields specified in the record output table will be written to the QDF. Field tables and RETRIEVE statements are processed ahead of the record output table. Consequently, the record produced may be composed of data which has been restructured via a field table and passed appropriate retrieval criteria. Each entry in the record output table produces a separate QDF record, provided the retrieval criteria and conditional expressions are met.

The record output table is particularly suited for creating data subsets, especially if a hierarchical structure with variable length records is required. The following example shows how a file could be converted from a flat to a hierarchical structure:

```
FILE OLD-FILE
FDT AUTO-FDT
QDF NEW-FILE
RECORD OUTPUT TABLE.
  FIELDS "OWNER RECORD ", NAME,CITY,"VIRGINIA" IF NAME CHANGES.
  FIELDS " AUTO RECORD ", MODEL,YEAR,COLOR.
END.
RUN
```

Input file:

JEFF BOGNAR	ALEXANDRIA	FORD	1983 BROWN
CHET BRITT	WOODBIDGE	FORD	1990 GREEN
CHET BRITT	WOODBIDGE	GEO	1990 WHITE
CHI HO ISEMAN	ALEXANDRIA	MERCEDES	1991 RED
CHI HO ISEMAN	ALEXANDRIA	VOLKSWAGEN	1960 BLUE
CHI HO ISEMAN	ALEXANDRIA	DODGE	1981 BROWN
KYUNG SEC	ARLINGTON	ACURA	1987 RED

Resulting QDF:

OWNER RECORD	JEFF BOGNAR	ALEXANDRIA	VIRGINIA
AUTO RECORD	FORD	1983 BROWN	
OWNER RECORD	CHET BRITT	WOODBIDGE	VIRGINIA
AUTO RECORD	FORD	1990 GREEN	
AUTO RECORD	GEO	1990 WHITE	
OWNER RECORD	CHI HO ISEMAN	ALEXANDRIA	VIRGINIA
AUTO RECORD	MERCEDES	1991 RED	
AUTO RECORD	VOLKSWAGEN	1960 BLUE	
AUTO RECORD	DODGE	1981 BROWN	
OWNER RECORD	KYUNG SEC	ARLINGTON	VIRGINIA
AUTO RECORD	ACURA	1987 RED	

Normally, the only output available from a record output table is the QDF itself (a QDF statement should always be included). Graphic displays are not possible unless the QDT is forced to match the QDF records. One means of accomplishing this is through the use of the BUILD QDT block structure.

Section 5. USER TERMINAL PROCESSING PROCEDURES

This section describes GIPSY statements which were implemented in order to interface with specific software developed for specific terminals for the WWMCCS community.

5.1. WIS/CUC Early Product Workstation and Z-248 PC/AT GIPSYmate Interface.

The interface between GIPSY and the GIPSYmate package gives the user the capability to recall prestored slides and to have GIPSY overlay them with geographic data. The command that accomplishes this is:

DISPLAY [MAP [<map options>]] [ON] SLIDE [<slide name>].

If the MAP clause is ommitted, the specified slide will be displayed as stored on the workstation. If MAP is specified, the slide will be displayed and active symbols, tracks, and circles will be plotted on top of it, along with the classification and title lines. The <map options> are the same as discussed in section 3.4.1.1.

The <slide name> is the name of the slide stored on the workstation. It may be from 1 to 35 characters in length. If the <slide name> contains blanks or begins with a numeric, it must be enclosed in quotes.

5.2 MAGIC/GIPSY Interface.

This interface provides the user with the capability to perform host data retrievals from the MAGIC workstation. This interface satisfies most users needs to transfer GIPSY-generated data sets to the workstation environment for further processing.

Data can be downloaded from GIPSY to the MAGIC workstation through this interface while the user is logged on the MAGIC workstation. GIPSY will transfer the proper data set to the workstation.

If the command is issued while the user is in the GEOMOD or GDRMOD subsystem of GIPSY, a QDT and a QDF data set will be sent to the workstation. If the command is issued while the user is in the DISPLA subsystem of GIPSY, a GDS data set is sent.

THIS PAGE INTENTIONALLY LEFT BLANK

Section 6. GIPSY's GENERALIZED DATA REPORTS

The purpose of this section is to provide a detailed description of how to use GIPSY's Generalized Data Reports (GDR) module (GDRMOD) in an on-line, interactive environment as well as non-interactive mode. GDRMOD provides GIPSY users with the capability to produce formatted, textual reports and gives the users complete control over the format of their reports. In addition, it provides a very flexible, dynamic paging capability for each report produced. A report may be viewed sequentially page by page, selectively by issuing specific commands, or may be directed to an on-line printer.

6.1 Initiating the Generalized Data Reports Module

The Generalized Data Reports is a separate GIPSY module. It may be executed from either within GIPSY or as a subsystem of GIPSY.

The Generalized Data Reports module may be entered directly from system level by entering the command "GIPSYR." This process will cause GIPSY to be automatically initiated. GIPSY will respond with an initial log-on message. The GDR module is ready for input when the GIPSY prompt ">" appears. When the GIPSY command "DONE" is entered, the user will be returned to system level and the system prompt "*" appears.

The Generalized Data Reports module may be accessed via two GIPSY commands while the user is operating within GIPSY -- RUN TO GDR, and TRANSFER TO GDR. The RUN TO GDR command will cause GIPSY to read the data file specified by the user in the FILE statement (see section 3.2.4.1), retrieve the records specified by the retrieval criteria on the RETRIEVE statement, and output the resulting data on a temporary file called the Qualified Data File (QDF). When the retrieval process is complete, GIPSY will display a message showing the number of records read and retrieved. GIPSY then places the user in the GDR module and the prompt character ">" is displayed.

The TRANSFER TO GDR command allows the GIPSY user to exit the GIPSY module currently executing and enter the GDR module. If the transfer command is issued while the user is in GDR, nothing happens -- the GDR module is not exited and reentered.

6.2 Data Manipulation

The GDR module uses the data records which meet the retrieval criteria and have been saved in a temporary file called a QDF. The QDF and its associated Qualified Descriptor Table (QDT) are passed to GDR. The user may wish to save the QDF/QDT for subsequent GDR runs. This makes it possible to bypass selection of data from the master file if the data and retrieval have not changed.

6.2.1 Field Manipulation. The capability exists within GIPSY to modify the QDT to support new fields. The BUILD QDT structure can be used to further

expand the QDT to include GLOBAL, QUALIFY, or QDF extended fields as well as QDF overlapping fields.

The syntax for the BUILD QDT structure is as follows:

```
BUILD QDT.  
[<add type>].  
[<command to add field definition>].  
END.
```

6.2.1.1 Add Type. The command for the <add type> is the single word ADD followed by the type of structure that it is to be added to. The syntax for the structure is:

```
ADD TO { QDF  
        QUALIFY  
        GLOBAL } .
```

6.2.1.2 Command to Add Field Definition. The command line used to create a new field to augment an existing file is structured as follows:

```
<field name1>      { <defined fieldname> [, <type and length>] }  
                   { <starting position> , <type and length> } .  
  
<field name2>      { <defined fieldname> [, <type and length>] }  
                   { <starting position> , <type and length> } .  
.  
.  
.  
null line.
```

The ADD TO QDF will define QDF fields like ADD to FILE defines file fields. For a more complete discussion of the BUILD process, see section 3.2.5.1 (Describing Data File Records to GIPSY).

6.2.2 Record Manipulation. The capability exists to provide user control over the database field manipulation to affect the contents of the report. These capabilities are as follows:

- a. Database sort
- b. Data record qualification
- c. Data element manipulation

6.2.2.1 Database Sort. A database sort occurs prior to the generation of a report. This may be done with either a SORT command or a RESORT command.

6.2.2.1.1 SORT Command. The capability to sort the data retrieval exists within GIPSY. The syntax for creating a sorted QDF is:

`SORT ON <fieldname1> [ASCENDING
DESCENDING] [{ ; }] <fieldname2> ASCEND
DESCEND]..] .`

For a detailed explanation of this statement, see section 4.2 which discusses the data sorting capability.

6.2.2.1.2 RESORT Command. The RESORT command performs a sort of the user's QDF. This action is immediately taken upon acceptance of the command. The syntax for the RESORT command is:

`RESORT TO { FILE <cfid> } QDF ON <sort field list> .`

The optional TO directs the resorted QDF to be placed either in the original QDF or another specified file. If neither FILE nor QDF is specified, the resorted QDF is placed in a temporary file generated by the RESORT process.

The GIPSY standard sort field list is used for the <sort field list> which includes sort field priority and the direction of the sort.

6.2.2.2 Data Record Qualification. The capability exists to dynamically alter the selection of data prior to generating a report. This may be done with the QUALIFY command. The QUALIFY command initiates the structure that GIPSY uses to subset the QDF based upon user criteria. Only those QDF records that pass the qualify criteria will be passed to the GDR module. Each specified QUALIFY structure replaces the previously specified QUALIFY structure. The syntax for the QUALIFY command is:

`QUALIFY { [IF] <conditional expression>
 <logic table entry>
 ALL } .`

The keyword IF is optional and is added only for clarity and readability. The <conditional expression> may be an IF clause or a previously defined name from a logic table. See section 3.4.4 for further details on the QUALIFY statement.

The only fields that can have further overlapping definitions are straight "QDF" fields. QUALIFY, QDF extended, and GLOBAL fields cannot have overlapping subordinate definitions.

6.2.3 QDF Manipulation. There are several GIPSY structures available in the output processor. These structures are processed against the QDF and include:

QUALIFY - Provides a logical subset of the QDF. Refer to section 3.4.4 for further explanation.

FIELD TABLE QDF - Provides data manipulation against QDF fields. Refer to Appendix C for further explanation.

FIELD TABLE QUALIFY - Provides data manipulation against QDF fields.
Refer to Appendix C for further explanation.

FIELD TABLE CALL - Provides an immediately executable Field Table structure. Refer to Appendix C for further explanation.

6.3 Report Building

GIPSY produces a variety of formatted textual reports. The GDR module provides the user the means to adequately define to GIPSY the format of displays. Figure 7-1 is provided as a guide to the user. It should be referenced for further clarification as each section is referenced below.

6.3.1 Report Definition. The GDR module allows the user to describe the format of his/her reports to GIPSY rather than forcing a pre-determined default format for the user's reports. The format is described by means of an output table. In defining the report, the user may specify two sections: PARTS and the GROUPS. The syntax for defining the output table is:

OUTPUT TABLE [NAMED] <name>.

The output table is a block structure bounded by OUTPUT TABLE and END. This block structure only defines the format of the reports. The following sections define processes within the structure.

6.3.1.1 Parts Definition. Each report may have one or several parts to it. The user should consider a part as the lowest level of data to be collected under a single heading. The main components of a part are the header, body and the trailer.

6.3.1.1.1 Header Table. This component allows the user to identify the information that is to be replicated at the top of each page. The header table is an optional substructure within the output table and, as such, must be specified within the OUTPUT TABLE structure. The header table must be ended by either the END HEADER or END commands. The syntax for the HEADER TABLE command is:

HEADER TABLE [PART <part number>].
 <output element statements>
.
.
.
END.

6.3.1.1.2 Body Table. The body table allows the user to format the principal portion of the report. The body table is an optional substructure within the output table and, as such, must be specified within the OUTPUT TABLE structure. The body table must be ended by either END BODY or END commands.

```

1000FILE LIBRARY/GIPS'/TEST/AIRFIELD
1010FDT LIBRARY/GIPSY/TEST/AFD-FDT
1020INCLUDE ALL FIELDS.
1030SORT ON STATE-CODE, NAME.
1040RUN TO GDR.
1050BUILD QDT. ADD.
1060 STATE-NAME * A14.
1070 CONDITION * A4.
1080 FIRST-REC * L.
1090 ALPHA-INDEX * BI.
1100 EQUAL * L.
1110 ALPHA-LIST * A26.
1120 GT * L.
1130 PPAGE * A3.
1140 ALPHA-HOLD * A78.
1150 XPART * BI.
1160 XLINE * A80.
1170 XLINEPT * BI.
1180END.
1190BOOKFILE LIBRARY/GIPSY/TEST/BKAIRFLD
1200//
1210FIELD TABLE /ODF.
1220 STATE-NAME = "ALABAMA" IF STATE-CODE EQ "AL".
1230 STATE-NAME = "MISSISSIPPI" IF STATE-CODE EQ "MS".
1240 STATE-NAME = "SOUTH CAROLINA" IF STATE-CODE EQ "SC".
1250 STATE-NAME = "GEORGIA" IF STATE-CODE EQ "GA".
1260 STATE-NAME = "FLORIDA" IF STATE-CODE EQ "FL".
1270 CONDITION = "GOOD" IF COND-CODE EQ "G".
1280 CONDITION = "FAIR" IF COND-CODE EQ "F".
1290 CONDITION = "POOR" IF COND-CODE EQ "P".
1300END.
1310//
1320FIELD TABLE /ODF FOR INITIAL.
1330 FIRST-REC = TRUE.

```

Figure 6-1. Sample PCS for BOOK Named BKAIRFIELD (Part 1 of 5)

```

1340 ALPHA-HOLD = " ".
1350 ALPHA-LIST = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
1360END.
1370//
1380FIELD ROUTINE OUTPUT NAMED INDEX-PAGE.
1390 FIELD TABLE REPEAT VARYING ALPHA-INDEX FROM 1 TO 26 OR UNTIL EQUAL.
1400 $COMPARE-STRING(XLINE,1,1,ALPHA-LIST,ALPHA-INDEX,1,EQUAL,GT).
1410 END.
1420 ALPHA-INDEX = ((ALPHA-INDEX - 1) * 3 + 1).
1430 PPAGE = $PAGE-NUMBER(5).
1440 $MOVE-STRING (ALPHA-HOLD,ALPHA-INDEX,3,PPAGE,1,3).
1450END.
1460//
1470FIELD ROUTINE OUTPUT NAMED OUTPUT-INDEX.
1480 FIELD TABLE REPEAT VARYING XPART FROM 1 TO 4.
1490 XLINE = " ".
1500 XLINE = STATE-XLINE.
1510 ALPHA-INDEX = ((XPART - 1) * 21 + 1).
1520 XLINEPT = 29.
1530//
1540 FIELD TABLE REPEAT VARYING ALPHA-INDEX FROM ALPHA-INDEX
      TO (ALPHA-INDEX + 19) BY 3 OR UNTIL ALPHA-INDEX GE 72.
1560 $MOVE-STRING(XLINE,XLINEPT,3,ALPHA-HOLD,ALPHA-INDEX,3).
1570 ADD 7 TO XLINEPT.
1580 END.
1590//
1600 LINE/XPART XLINE.
1610 END.
1611 ALPHA-HOLD = " ".
1612END.
1613//
1620OUTPUT TABLE NAMED AIRFLD.
1630//
1640GROUP INDEX = 1:2:3:4, RUNWAYS = 5.

```

Figure 6-1. Sample PCS for BOOK Named BKAIRFIELD (Part 2 of 5)

```

1650TOTAL PAGES.
1660//
1670HEADER TABLE PART 1.
1680 LINE "PAGE":$PAGE-NUMBER(1) PIC "Z,ZZZ" COL 65
1690 LEFT JUSTIFIED TEXT COMPRESSED.
1700 SPACE 1.
1710 LINE "AIRFIELDS ALPHABETICAL INDEX" CENTERED.
1720 LINE "A - G" CENTERED.
1730 SPACE 2.
1740 LINE "A" B C D E F G COL 30.
1750 SPACE 2.
1760END HEADER.
1770//
1780HEADER TABLE PART 2.
1790 LINE "PAGE":$PAGE-NUMBER(2) PIC "Z,ZZZ" COL 65
1800 LEFT JUSTIFIED TEXT COMPRESSED.
1810 SPACE 1.
1820 LINE "AIRFIELDS ALPHABETICAL INDEX" CENTERED.
1830 LINE "H - N" CENTERED.
1840 SPACE 2.
1850 LINE "H" I J K L M N COL 30.
1860 SPACE 2.
1870END HEADER.
1880//
1890HEADER TABLE PART 3.
1900 LINE "PAGE":$PAGE-NUMBER(3) PIC "Z,ZZZ" COL 65
1910 LEFT JUSTIFIED TEXT COMPRESSED.
1920 SPACE 1.
1930 LINE "AIRFIELDS ALPHABETICAL INDEX" CENTERED.
1940 LINE "O - U" CENTERED.
1950 SPACE 2.
1960 LINE "O" P Q R S T U COL 30.
1970 SPACE 2.
1980END HEADER.

```

Figure 6-1. Sample PCS for BOOK Named BKAIRFIELD (Part 3 of 5)


```

1990//
2000HEADER TABLE PART 4.
2010 LINE "PAGE":$PAGE-NUMBER(4) PIC "Z,ZZZ" COL 65
2020 LEFT JUSTIFIED TEXT COMPRESSED.
2030 SPACE 1.
2040 LINE "AIRFIELDS ALPHABETICAL INDEX" CENTERED.
2050 LINE "V - Z" CENTERED.
2060 SPACE 2.
2070 LINE "V" W X Y Z" COL 30.
2080 SPACE 2.
2090END HEADER.
2100//
2110HEADER TABLE PART 5.
2120 LINE $PRINT-DATE LEFT JUSTIFIED, "PAGE":$PAGE-INKEEP(5) PIC "Z,ZZZ":
2130 " OF ":$TOTAL-PAGES(5) PIC "Z,ZZZ" RIGHT JUSTIFIED TEXT COMPRESSED.
2140 SPACE 1.
2150 LINE "RUNWAY INFORMATION" CENTERED.
2160 SPACE 2.
2170 LINE "NAME" COL 28, "SURFACE" COL 51, "CONDITION" COL 60,
2180 "LENGTH" COL 72.
2190 SPACE 1.
2200END HEADER.
2210//
2220BODY TABLE PART 5.
2230//
2240 LINE STATE-NAME:" (CONTINUED)" LEFT JUSTIFIED
2250 IF $LINES-LEFT(5) EQ 0.
2260//
2285 EJECT IF STATE-CODE CHANGES AND NOT FIRST-REC.
2290 FIELD TABLE WHEN STATE-CODE CHANGES.
2310 LINE STATE-NAME LEFT JUSTIFIED.
2320 END.
2330//
2340 SPACE 1 IF NAME(1/1) CHANGES.

```

Figure 6-1. Sample PCS for BOOK Named BKAIRFIELD (Part 4 of 5)

```

2350 LINE NAME COL 17, SURFACE COL 51, CONDITION COL 63,
2360 LENGTH PIC "Z2,Z22"; "FT" COL 71.
2370//
2371 $$INDEX-PAGE IF NAME(1/1) CHANGES.
2374 $$OUTPUT-INDEX IF STATE-CODE COMPLETE.
2380END BODY.
2390END.
2400//
2410BUILD OUTPUT INDEX AIRFLD TO BKAIRFLD.

```

*

Figure 6-1. Sample PCS for BOOK Named BKAIRFIELD (Part 5 of 5)

The BODY TABLE command may also have an optional FOR clause following the <part assignment>. The syntax for the BODY TABLE command is:

```
BODY TABLE [PART <part number>] [FOR <for clause>].  
  <output element statements>  
.  
.  
.  
END.
```

where <for clause> specifies when this structure will be executed and may be one of the following:

```
INITIAL  (when the first record is processed)  
FINAL    (after the last record has been processed)
```

when the FOR clause is not specified, processing occurs for each record.

6.3.1.1.3 Trailer Table. This component allows the user to identify the information that is to be replicated at the bottom of each page. The trailer table is an optional substructure within the output table and, as such, must be specified within the OUTPUT TABLE structure. The trailer table must be ended by either the END TRAILER or END command. The syntax for the TRAILER TABLE command is:

```
TRAILER TABLE [PART <part number>].  
  <output element statements>  
.  
.  
.  
END.
```

6.3.1.1.4 Output Element Statements. All of the executable statements available within the FIELD TABLE QDF structure are also available to the <output element statements>. There are three more statements, though totally unique which may be used in formatting a report. They are as follows:

```
LINE    - Builds and outputs one line of the report  
  
SPACE   - Outputs line feeds (also known as spaces or carriage returns)  
          to the report  
  
EJECT   - Outputs page ejects (also known as form feeds or new pages).
```

The syntax and use of each of these statements is outlined below along with several other examples.

6.3.1.1.4.1 The LINE Statement. The LINE statement builds and outputs one line to the report. Output elements are separated by commas and a colon concatenates data elements together for use as a single output element. A

data element refers to any QDF, QDF extended, QUALIFY, GLOBAL fields, and/or an alphanumeric literal. This can also be the result from the execution of a user subroutine (see section 6.4.5.3). The complete syntax of the LINE statement is:

```
LINE [/<part number>] <line number> [IF <conditional expression>].
```

where <part number> is the part in which the line is to be output. This field, if specified, will override a default part assignment made in the subordinate structure, as in "HEADER TABLE PART 2". If no part assignment is given in the line statement, then <part number> defaults to whatever was set in the subordinate structure. If, on the other hand, no part assignment is given at all, then <part number> defaults to one.

where <line specification> is one or more complete output elements, including placement (COL, CENTERED, etc.) and/or picture mask (PIC). Each output element on a line must be separated by commas and be self-contained.

Data elements used to build output elements include any combination of the following:

```
<field>  
"<alphanumeric literal>"  
$<user subroutine>  
$$<field routine>  
(<arithmetic expression>)
```

Each output element may be placed on the report with any of the following phrases:

```
COL <column number>  
COL + <column number relative to end of last column>  
RIGHT JUSTIFIED [<placement option>]  
LEFT JUSTIFIED [<placement option>]  
CENTERED [<placement option>]
```

where the various <placement options> are:

WITHIN followed by the number of columns in which to place this entire element (including before and after blanks). If the number of columns is not specified, this will default to the whole page. This may cause over-lapping elements; keep in mind that the last element built takes priority.

TEXT WITHIN followed by the number of columns in which to place this element (excluding before and after blanks). If the number of columns is not specified, this will default to the whole page. This may cause over-lapping elements; keep in mind that the last element built takes

priority.

TEXT COMPRESSED WITHIN followed by the number of columns in which to place this element. All repeating blanks within the element will be compressed into one blank. If the number of columns is not specified, this will default to the whole page. This may cause overlapping elements; keep in mind that the last element placed takes priority.

TEXT PACKED WITHIN followed by the number of columns in which to place this element. All blanks in the element will be eliminated completely. If the number of columns is not specified, this will default to the whole page. This may cause overlapping elements; keep in mind that the last element placed takes priority.

The following examples will outline the various options available within the LINE statement. Each example will utilize the sample QDF record below and its supporting BUILD QDT structure:

ARMY FORT HUACHUCA TANKS 25517092

BUILD QDT.

ADD.

SERVICE NAME 1 A10.

BASE-NAME 11 A17.

EQUIP-NAME 28 A6.

NBR-EQUIP 34 I5.

READY-PCNT 39 I3.

END.

EXAMPLE 1:

LINE "THIS IS A REPORT FOR ":SERVICE-NAME.

1	2	3	4	5	6	7
123456789012345678901234567890123456789012345678901234567890						

THIS IS A REPORT FOR ARMY

The literal "THIS IS A REPORT FOR" and the contents of the QDF field SERVICE-NAME are concatenated together to form a single output element. The element was output starting in column 1 by default as no column position was given in the statement.

EXAMPLE 2:

LINE "THIS IS A REPORT FOR ":SERVICE-NAME COL 15.

1	2	3	4	5	6	7
1234567890123456789012345678901234567890123456789012345678901234567890						

THIS IS A REPORT FOR ARMY

The output element "THIS IS A REPORT FOR ARMY" was built and output starting in column 15. The option to note in this example is "COL". It specifies the starting position (in this case, column 15) for the entire output element. Remember that the colon (":") separates data elements to be treated as a single output element.

EXAMPLE 3:

LINE "THIS IS A REPORT FOR "COL 10, SERVICE-NAME COL 35.

1	2	3	4	5	6	7
1234567890123456789012345678901234567890123456789012345678901234567890						

THIS IS A REPORT FOR ARMY

The literal "THIS IS A REPORT FOR " was output starting in column 10. The contents of the field SERVICE-NAME were output starting in column 35. Remember the comma (",") separates output elements. Each output element was given its own starting position by using the key word "COL" followed by a number for each element.

EXAMPLE 4:

LINE "THIS IS A REPORT FOR "COL 10, SERVICE-NAME COL + 2.

1	2	3	4	5	6	7
1234567890123456789012345678901234567890123456789012345678901234567890						

THIS IS A REPORT FOR ARMY

The literal "THIS IS A REPORT FOR" was output starting in column 10. The contents of the field SERVICE-NAME was output with two blanks separating it from the end of the literal. An internal "end of last element" count is kept for the user at all times. The plus sign ("+") following the key word "COL" signifies that this count be used to compute the starting position of the output element. The count is always kept one column past the end of the last element so "COL + 0" places the element flush against the last one and "COL + 3" puts three blanks between the elements.

EXAMPLE 5:

LINE "THIS IS A REPORT FOR ":SERVICE-NAME CENTERED.

1	2	3	4	5	6	7
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

THIS IS A REPORT FOR ARMY

The output element "THIS IS A REPORT FOR ARMY" was built and centered on the page. The key word "CENTERED" dictates centering. When CENTERED is not followed by any of the centering options (discussed next), the element is centered on the page.

EXAMPLE 6:

LINE "THIS IS A REPORT FOR " CENTERED WITHIN 30, SERVICE-NAME COL 15.

1	2	3	4	5	6	7
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

THIS IS A REPORT FOR ARMY

The output element "THIS IS A REPORT FOR" was placed on the report centered within the first 30 columns. The key words "CENTERED WITHIN", followed by a number, dictate that the element is to be centered within a column span. The "end of last element" counter is used to determine the starting position of the column span since no column was specified. The counter starts at 1 for every new line. The element was then centered within columns 1 and 30. The output element, the content of the field SERVICE-NAME, was placed starting in column 35.

EXAMPLE 7:

LINE "THIS IS A REPORT FOR " COL 15 CENTERED WITHIN 30, SERVICE-NAME COL 50.

1	2	3	4	5	6	7
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

THIS IS A REPORT FOR ARMY

The literal "THIS IS A REPORT FOR" was centered within 30 columns and the beginning of the span was set to 15. Therefore, the element was centered within columns 15 through 45. The content of the field SERVICE-NAME was output starting at column 50.

EXAMPLE 8:

LINE "THIS IS A REPORT FOR ", SERVICE-NAME CENTERED WITHIN 20.

THIS IS A REPORT FOR ARMY

EXAMPLE 9:

THIS IS A REPORT FOR ARMY

EXAMPLE 10:

FORT HUACHUCA

6-15

to be 17 characters long and the entire field, including blanks, was justified on the page.

EXAMPLE 11:

LINE "THIS IS A REPORT FOR " COL 5, SERVICE-NAME COL + 2, BASE-NAME RIGHT JUSTIFIED TEXT.

1	2	3	4	5	6	7
123456789012345678901234567890123456789012345678901234567890						

THIS IS A REPORT FOR ARMY

FORT HUACHUCA

In this example, the content of the field BASE-NAME was right justified without trailing blanks. Again, the key word "TEXT" dictates that this happens.

EXAMPLE 12:

LINE "THIS IS A REPORT FOR " COL 5, SERVICE-NAME COL + 2, "EQUIPMENT LOCATED AT ":BASE-NAME RIGHT JUSTIFIED TEXT.

1	2	3	4	5	6	7
123456789012345678901234567890123456789012345678901234567890						

THIS IS A REPORT FOR ARMY

EQUIPMENT LOCATED AT FORT HUACHUCA

Here, the output element "EQUIPMENT LOCATED AT FORT HUACHUCA" was built and right-justified on the page without the extra blanks in the field BASE-NAME. Again, this is due to the key word TEXT.

EXAMPLE 13:

LINE "SERVICE ":SERVICE-NAME LEFT JUSTIFIED TEXT, "STATUS REPORT " CENTERED, "BASED ":BASE-NAME RIGHT JUSTIFIED TEXT.

1	2	3	4	5	6	7
123456789012345678901234567890123456789012345678901234567890						

SERVICE ARMY

STATUS REPORT

BASE FORT HUACHUCA

In this example, no columns or column spans were given following the key words "LEFT JUSTIFIED", "CENTERED", and "RIGHT JUSTIFIED". Without specific starting positions, these placement options default to use the entire page.

EXAMPLE 14:

LINE "THIS IS A REPORT FOR ":SERVICE-NAME LEFT JUSTIFIED TEXT, "STATUS REPORT" CENTERED, "EQUIPMENT LOCATED AT ":BASE-NAME RIGHT JUSTIFIED TEXT.

1 2 3 4 5 6 7
123456789012345678901234567890123456789012345678901234567890

THIS IS A REPORT FOR ARMY STATUS EQUIPMENT LOCATED AT FORT HUACHUCA

This line has a problem. The output processor built this line exactly the way the user asked it to. First, the output element "THIS IS A REPORT FOR ARMY" was built and placed as requested. Second, the output element "STATUS REPORT" was built and placed and third, the output element "EQUIPMENT LOCATED AT FORT HUACHUCA" was built and placed. When the third element was placed, it overwrote part of the second element. This is a possibility that should be kept in mind when building output elements.

EXAMPLE 15:

LINE "A USER WOULD PROBABLY NEVER DO THIS" CENTERED TEXT
COMPRESSED.

1 2 3 4 5 6 7
123456789012345678901234567890123456789012345678901234567890

A USER WOULD PROBABLY NEVER DO THIS

While the example appears to be a little ridiculous, the concept being shown is not. The key words "CENTERED TEXT COMPRESSED" dictate that all repetitive blanks be reduced down to only a single blank. Just because the example says that "A USER WOULD NEVER DO THIS", does not mean that a need for the "COMPRESSED" option would never exist.

EXAMPLE 16:

LINE "STATUS REPORT FOR ":SERVICE-NAME:" AT ":BASE-NAMED CENTERED TEXT
COMPRESSED.

1 2 3 4 5 6 7
123456789012345678901234567890123456789012345678901234567890

STATUS REPORT FOR AT FORT HUACHUCA

Remember that SERVICE-NAME is a 10 character field as defined in the QDT. In our example database record, the content of that field is "ARMY".

The "COMPRESSED" option removed the additional spaces between the words "ARMY" and "AT". The key words "TEXT" meant that the blanks following the contents of BASE-NAME were not included in the centering; "COMPRESSED" alone would have compressed these extra blanks down to one blank instead of ignoring them.

```

LINE "A      USER      WOULD      PROBABLY      NEVER      DO      THIS"
LEFT JUSTIFIED TEXT COMPRESSED, "WOULD      HE?" RIGHT JUSTIFIED
TEXT COMPRESSED.

```

A USER WOULD PROBABLY NEVER DO THIS WOULD HE?

EXAMPLE 18:

AUSERWOULDPROBABLYNEVERDOTHIS ATLEAST IDON'TTHINKSO

EXAMPLE 19:

ARMY	TANKS	TOTAL EQUIPMENT	25,517
------	-------	-----------------	--------

6-18

COBOL picture mask. Below is an example which illustrates the PIC clause, the beginning number, the picture mask held against it, and the resulting number:

<u>NUMBER</u>	<u>PICTURE MASK</u>	<u>RESULTS</u>
25517	ZZ,ZZZ,ZZ9	25,517
33	ZZ,ZZZ,ZZ9	33
0	ZZ9	0
1337295	ZZ,ZZZ.Z9	13,372.95
0	ZZ,ZZZ.99	.00
0	ZZ,ZZ9.99	0.00
-1	ZZ,ZZ9.99	-1.00
1.792	ZZ9.9	1.8
1.792	ZZ9.99	1.79
1.555	ZZ9.9	1.6
1.555	ZZ9.99	1.56
-1.558	ZZ9.99	-1.56
14785	ZZ9.99	147.85
147.85	++,++9.99	+147.85
147.85	++9.99	*****
-147.85	++,++9.99	-147.85
8765.43	\$ZZ,ZZZ,99	\$8,765.43
.20	\$ZZ,ZZZ.99	\$.20
8765.43	\$\$\$,\$\$\$,99	\$8,765.43
.20	\$\$\$,\$\$\$9.99	\$0.20
-.20	\$\$\$,\$\$\$9.99	\$-0.20

If the resulting number will not fit into the mask, a string of asterisks is returned. If the number has three digits after the decimal point and the mask shows only two, the resulting number will be rounded to the nearest place. If a number is negative, the minus sign ("-") will always be forced onto the resulting number; if the minus sign doesn't fit, a string of asterisks is returned.

6.3.1.1.4.2 The SPACE Statement. The SPACE statement allows the user to force line feeds to the page. The complete syntax of this statement is:

SPACE [/<part number>] <space parameter> [IF <conditional expression>].

where <part number> is the part in which the line is to be output. This field, if specified, will override a default part assignment made in the subordinate structure, as in "HEADER TABLE PART 2". If no part assignment is given in the SPACE statement, then <part number> defaults to whatever was set in the subordinate structure. If, on the other hand, no part assignment was given at all, then <part number> defaults to one.

where <space parameter> is the number of line feeds to perform. This may be a numeric literal, a mathematical expression (enclosed in parentheses), or a field.

The following examples illustrate some uses of the SPACE statement.

EXAMPLE 1:

```
HEADER TABLE PART 3
  LINE TITLE CENTERED.
  SPACE 2.
END.
```

In this example, the content of the field TITLE will be centered on the page for PART 3. Two blank lines will follow, separating the title from the rest of the page. Notice that both the LINE and the SPACE statement use the default part assignment of 3 set in the subordinate structure opening sentence.

EXAMPLE 2:

```
BODY TABLE PART 5.
  LINE/6 "THIS GOES ON PART 6".
  SPACE/6 1.
  LINE "THIS GOES ON PART 5".
  SPACE 1.
END.
```

Although the default part assignment for this BODY TABLE is 5, the literal in the first LINE statement and the line feed from the SPACE statement following it will be output for PART 6. The next LINE and SPACE statement will use the default part assignment of 5.

EXAMPLE 3:

```
TRAILER TABLE.
  SPACE 3.
  LINE "TOTAL AVERAGE-   ':TAVG COL 20.
END.
```

Here, no part assignment is specified, so <part number> automatically defaults to one for both the LINE and the SPACE statement.

6.3.1.1.4.3 The EJECT Statement. The EJECT statement allows the user to force page breaks in a report. The complete syntax of this statement is:

```
EJECT [/<part number>] [IF <conditional expression>].
```

where <part number> is the part in which the page break is to occur. This field, if specified, will override a default part assignment made in the subordinate structure, as in "HEADER TABLE PART 2". If no part assignment is given in the EJECT statement, then <part number> defaults to whatever was set in the subordinate structure. If, on the other hand, no part assignment is given at all, then <part number> defaults to one.

Below are a few examples of the EJECT statement.

EXAMPLE 1:

```
TRAILER TABLE.  
  LINE "MONTHLY TOTALS: ":COL 20.  
  EJECT  
END.
```

Here we have no part number specified, so the default value of one will apply for both the LINE and EJECT statements. After the literal and the contents of TOTAL is written out, a page break will occur every time this TRAILER TABLE is executed.

EXAMPLE 2:

```
BODY TABLE PART 4.  
  LINE STATE-NAME COL 10, AIRFLD COL 30.  
  EJECT/2 IF STATE-NAME CHANGES AND NOT FIRST-REC.  
  LINE/2 AIRFLD COL 10, CONDITION COL 30.  
END.
```

In this example, PART 4 will contain a list of states and associated airfields. A page break will occur for PART 2 every time STATE-NAME changes (and not the first record). This will give you a list of airfields and their condition, for PART 2, broken down by state name. Notice that the "/2" part assignments override the subordinate structure opening sentence part assignment.

6.3.1.1.5 The Output Table Subroutines. This section defines the GIPSY subroutines available within the output table substructure. The subroutine name is preceded by a dollar sign "\$" and arguments that either pass or receive data from the subroutine are field names defined in the FDT or QDT.

6.3.1.1.5.1 \$BODY-LINE. Subroutine \$BODY-LINE will return the number of the current body line count of a page being processed. This number is useful in conditional expressions. The syntax for this statement is:

\$BODY-LINE (<part number>)

where <part number> is the specific part for which you want the body line count.

The following example will advance the print line two spaces if the current body line count for part 1 is not equal to line 1:

SPACE 2 IF \$BODY-LINE(1) NE 1.

6.3.1.1.5.2 \$BODY-PART. Subroutine \$BODY-PART will return the part number currently being processed. The syntax for this statement is:

\$BODY-PART

The following example illustrates this subroutine in a conditional expression:

```
GROUP ALPHA=1, INDEX=2, SUMMARY=3:4:5.  
HEADER TABLE.  
    LINE "SUMMARY OF FINDINGS" CENTERED IF  
        $BODY-PART BT 3/5.
```

The literal "SUMMARY OF FINDINGS" will be centered in the heading whenever the current body part is part of the group called SUMMARY.

6.3.1.1.5.3 \$LINES-LEFT. Subroutine \$LINES-LEFT will return the number of body lines left on the current page. The syntax for this statement is:

\$LINES-LEFT (<part number>)

where <part number> is the specific part for which a body line count of lines left on the page is done.

The following example illustrates this subroutine in a conditional expression:

```
EJECT IF $LINES-LEFT(1) LT 5 AND $LINES-LEFT(1) NE 0.
```

A page eject will occur for PART 1 only if the number of lines left on the page is between zero and five.

6.3.1.1.5.4 \$PAGE-NUMBER. The subroutine \$PAGE-NUMBER will output the current page number of the specified part. The syntax for this statement is:

\$PAGE-NUMBER (<page number>)

where <page number> is the specific part for which the page number is to be output.

In the following example, the page number of each page of PART 2 will be right-justified in the heading for PART 2 along with the literal "PAGE":

```
HEADER TABLE PART 2.  
    LINE "PAGE":$PAGE-NUMBER(2) RIGHT JUSTIFIED.
```

6.3.1.1.5.5 \$PRINT-DATE. Subroutine \$PRINT-DATE will output the date the report is generated on as an output element to a specified output location. The syntax for this statement is:

\$PRINT-DATE

In the following example, the current date will be left justified in the heading for PART 3. The format for the date is dd/mm/yy (i.e., 11/MAY/85):

HEADER TABLE PART 3.

LINE \$PRINT-DATE LEFT JUSTIFIED.

6.3.1.1.5.6 \$PRINT-TIME. Subroutine \$PRINT-TIME will output the current time as an output element to the specified output location. The syntax for this statement is:

\$PRINT-TIME

The following example will left-justify the current time in the heading for PART 3. The format for the time is the standard military format (HH:MM):

HEADER TABLE PART 3.

LINE \$PRINT-TIME LEFT JUSTIFIED.

6.3.1.1.5.7 \$TOTAL-PAGES. Subroutine \$TOTAL-PAGES returns the total number of pages in a report. The syntax for this statement is:

\$TOTAL-PAGES (<part number>)

where <part number> is the specific part for which a total page count is desired.

6.3.1.2 Group Definition. As previously discussed the user has broken the report into parts. The user may wish to assemble one or more parts into a meaningful, manageable display. This may be done by specifying the GROUP command and assigning it the part numbers to be viewed. The syntax for the GROUP statement is:

GROUP <group name> = <part number> [:<part number>][, <group name>].

Where <group name> is the name assigned to the group by the user and <part number> is the specific part to be associated with that GROUP.

6.3.2 Report Preparation. The BUILD OUTPUT Statement initiates the operation of report generation. Prior to this time, the output table commands have just specified the report format and criteria. This process involves the retrieving of records from the QDF and passing the records through the FIELD TABLE QDF and the specified output table structure.

The finished report is then output to the BOOK location defined. The syntax for the BUILD OUTPUT statement is:

BUILD OUTPUT [NAMED] <name> TO <book name>.

where <name> is the 1-36 character name assigned to the output table defining the format of the report.

The BOOK file must be created before entering the GDR module. If the user enters from another GIPSY module, the BOOK file must be created before

entering GIPSY. The creation of a file is done at the Honeywell system level in the ACCESS subsystem. The BOOK file must be a random file. The example below shows the "short hand" creation of a random file 100 little links (llinks) in size and being given no specific permission:

```
ACCE CF, <cat/file>, B/100/,MODE/RANDOM/
```

where <cat/file> is the standard Honeywell character string pointing to the file the user wishes to create.

Once the BOOK file has been created, the end-of-file mark must be put on the file, to "un-null" it.

This is done at the Honeywell system level with the following command:

```
ERASE <cat/file>
```

where <cat/file> is the standard Honeywell character string pointing to the file the user created.

6.4 Report Viewing. After creating a BOOK, the user has the option of reviewing the BOOK's contents in detail. This capability is available through the DISPLAY BOOK command which can be utilized in either interactive or non-interactive mode.

6.4.1. Interactive Book Review. In interactive mode, display processing is very similar to a GIPSY block structure where a display command may be entered into the terminal in a conversational manner. At any time during this session, the user has the option of issuing another display command or ending the current review. The syntax for the DISPLAY BOOK command in interactive mode is:

```
DISPLAY BOOK <book name>      {  *  }  
                                { INTERACTIVE }  
                                .  
    <display command 1>  
    .  
    .  
    .  
    <display command n>  
END.
```

Where <book name> is the BOOK to be displayed and <display command> can be a group name, part number, and/or page specification. A detailed discussion of all available display commands can be found in section 6.4.3.

Before viewing a BOOK, it is essential that the user understand how a BOOK is structured. A thorough comprehension of these concepts enables the user to quickly and easily locate a given section of the whole BOOK for subsequent review and/or output.

An important concept to understand when displaying a BOOK is that of the viewing window, sometimes called the viewport. A viewing window can be thought of as a picture frame which encloses the area that you, the user, specify. While you can look at any one piece of the whole picture in detail, it must be physically within the boundaries of the picture frame for you to see it. During display processing, GDR looks at BOOKs the same way. The DISPLAY BOOK command is, essentially, a viewing window specification which, for GDR purposes, will always be a <group name>. For interactive BOOK viewing, this may be specified as a <display command>. A special case of this concept is when the user enters "ALL GROUPS" as a display command. GDR processes this command in the same manner as a normal <group name> except that the viewing window is expanded to treat all groups as a single, large GROUP. Subsequent BOOK review will result in all groups being viewed sequentially, and all their subordinate parts being viewed in the logical order that they were referred to in the GROUP statement (see section 6.3.1.2).

To interactively review the BOOK shown in figure 6-2, the user would enter "DISPLAY BOOK BKAIRFLD *." This command not only instructs GDR to view the BOOK interactively, it also uses default values to specify a viewing window. Since GDR establishes a default value of 1 for GROUP, PAGE, and PART, the resultant display will be INDEX, PAGE1, PART1 as shown in figure 6-2, Part 1 of 4.

To completely understand the viewing process, a discussion of PAGEs and PARTs is required. A PART is the smallest physical subdivision of a BOOK. It is a subordinate structure which can be incorporated into a logical grouping called a GROUP. Each GROUP must have at least one PART. A PAGE can best be thought of in terms of the screen of a computer terminal. Each PART must have at least one PAGE and when the user is viewing a BOOK, GDR displays it in terms of PAGEs. Consequently, in order to step through the remaining parts of the GROUP named INDEX, the user should continue to enter "/NP". This command instructs GDR to advance to the next PART, leaving the PAGE specification at the same value. For further information on PART structures, refer to section 6.3.1.1 and a full discussion of all display commands can be found in section 6.4.3. However, from this point on, it is the user's discretion to continue entering display commands or to terminate the current session with the END command.

6.4.2 Non-Interactive Book Review. Non-interactive display processing is useful when reviewing exceptionally large BOOKs. It permits the user to display any number of specified PAGEs from any number of specified GROUPs and/or PARTs. PAGEs will be displayed in the order specified by simply responding with a carriage return <CR> after each PAGE is displayed. GDR will respond with the prompt character ">" when all display commands have been processed. The user should be reminded that BOOKs to be reviewed non-interactively, must have been described using GROUP names with all PARTs (i.e. each PART must be associated with a GROUP name).

AIRFIELDS ALPHABETICAL INDEX A - G

	A	B	C	D	E	F	G
ALABAMA	1	1	1	1		1	1
FLORIDA	2	2	2	3	3	3	3
GEORGIA	5	5	5	6	6	6	6
MISSISSIPPI	8	8	8			9	9
SOUTH CAROLINA	10	10	10	10		10	11

Figure 6-2. Display of GROUP Named INDEX (Part 1 of 4)

AIRFIELDS ALPHABETICAL INDEX
H - N

	H	I	J	K	L	M	N
ALABAMA	1					1	
FLORIDA	3		3	3	3	3	4
GEORGIA	6		7		7	7	7
MISSISSIPPI	9		9	9	9	9	
SOUTH CAROLINA	11					11	

Figure 6-2. Display of GROUP Named INDEX (Part 2 of 4)

AIRFIELDS ALPHABETICAL INDEX O - U

	O	P	Q	R	S	T	U
ALABAMA				2	2	2	
FLORIDA	4	2	4		4	5	
GEORGIA		4	7	7	7	8	
MISSISSIPPI	9	10		10	10		
SOUTH CAROLINA	11				11		

Figure 6-2. Display of GROUP Named INDEX (Part 3 of 4)

AIRFIELDS ALPHABETICAL INDEX V - Z

	V	W	X	Y	Z
ALABAMA					
FLORIDA					
GEORGIA					
MISSISSIPPI					
SOUTH CAROLINA					
	5	2			
	8	5			
	10	8			
		11			

Figure 6-2. Display of GROUP Named INDEX (Part 4 of 4)

The syntax for the non-interactive DISPLAY BOOK command is:

```
DISPLAY BOOK <book name> <group name1> <display command>  
          [, <group name2> <display command> ,....] .
```

Where <book name> is the BOOK to be displayed, <group name> refers to the GROUP the user wishes to review, and <display command> can be a PAGE and/or a PART command.

Referring to the BOOK in our previous example but using a different GROUP as shown in figure 6-3, the user wishing a non-interactive viewing could enter "DISPLAY BOOK BKAIRFLD RUNWAY FP, Page 4, LP.". This command will cause three pages of the GROUP named RUNWAY to be displayed. Since no PART specifications were made, the first, fourth, and last pages of PART1 will be shown as described previously using a carriage return <CR> to signal page advance. However, if a <display command> is entered at the end of a display rather than a carriage return <CR>, the processing mode changes to interactive. The user may then input any and all display commands until, wishing to terminate the session, entering the END command. This ends display processing and the GDR prompt character ">" appears. Therefore, the user is cautioned when using this option since the transfer to interactive processing is a one-way street.

At the time of transfer, any unfinished, non-interactive commands are lost and must be reentered.

6.4.3 Book Review Display Commands. The majority of display commands available for display processing, are in reference to PAGES, PARTs, or both. This is only natural since you, as a user, will view a BOOK through its PARTs and each PART is displayed on your screen as one or more PAGES. All PAGE and PART commands may be entered individually or in PAGE/PART pairs where the "/" is a delimiter signifying a PART command. The user should be cautioned that when entering a PAGE/PART command pair, to be sure that the values specified are both valid and accurate. The remainder of the display commands are either GROUP specifications or commands that are meaningful in terms of GDR processing. The following is a list and description of the various display commands available during display processing:

ALL GROUPS	displays all PAGES of all PARTs of all GROUPs, beginning with PAGE1/PART1 of the first GROUP declared in the GROUP statement. In the case where a <group name> is associated with more than one PART, all PARTs will be paged down in the logical sequence they were specified in.
ALL PAGES (AP)	permits the user to display all PAGES within the currently specified PART.

RUNWAY INFORMATION

NAME	SURFACE	CONDITION	LENGTH
ALABAMA			
ALBANYVILLE MUNICIPAL	ASPHALT	GOOD	4,800FT
ANNISTON/CALHOUN COUNTY	ASPHALT	FAIR	5,000FT
AUBURN OPELIKA ROBERT G PIT	ASPHALT	FAIR	4,000FT
BATES FIELD	ASPHALT	GOOD	8,527FT
BESSEMER	ASPHALT	GOOD	3,800FT
BIRMINGHAM MUNICIPAL	CONCRETE	GOOD	10,000FT
BREWTON MUNICIPAL	ASPHALT	GOOD	5,100FT
BROOKLEY	CONCRETE	GOOD	9,600FT
CAIRNS	ASPHALT	GOOD	5,000FT
DANIELLY FIELD	ASPHALT	GOOD	9,001FT
DOTHAN	ASPHALT	GOOD	8,495FT
FOLSOM FIELD	ASPHALT	FAIR	5,120FT
GADSDEN MUNICIPAL	CONCRETE	FAIR	6,800FT
GRAGG/MADE FIELD	ASPHALT	GOOD	4,000FT
HUNTSVILLE MADISON CO CARL	ASPHALT	GOOD	8,000FT
MARION COUNTY	ASPHALT	GOOD	7,000FT
MAXWELL AFB	ASPHALT	GOOD	7,000FT
MONROE COUNTY	ASPHALT	GOOD	6,000FT

Figure 6-3. Display of GROUP Named RUNWAY (Part 1 of 5)

20/AUG/85

PAGE 2 OF 11

RUNWAY INFORMATION

	NAME	SURFACE	CONDITION	LENGTH
ALABAMA	(CONTINUED)			
	MUSCLE SHOALS	ASPHALT	GOOD	6,693FT
	PRYOR FIELD	ASPHALT	GOOD	5,096FT
	REDSTONE AAF	ASPHALT	FAIR	7,300FT
	SHELBY COUNTY ST CLAIR COUNTY	ASPHALT ASPHALT	GOOD GOOD	3,800FT 4,200FT
FLORIDA	THOMAS C RUSSELL TROY MUNICIPAL TUSCALOOSA MUNICIPAL	ASPHALT CONCRETE ASPHALT	GOOD FAIR GOOD	3,550FT 4,995FT 6,499FT
	WETUMPKA MUNICIPAL	ASPHALT	GOOD	3,000FT
	ALBERT WHITTED	ASPHALT	GOOD	3,322FT
	BARTONJ MUNICIPAL BOB SIKES	ASPHALT ASPHALT	GOOD GOOD	5,000FT 8,000FT
	CAMP BLANDING AAF CAPE CANAVERAL AIR FORCE ST CECIL FIELD NAS CHARLOTTE COUNTY CRAIG MUNICIPAL	GRASS ASPHALT ASPHALT ASPHALT ASPHALT	GOOD GOOD GOOD POOR FAIR	3,200FT 10,000FT 12,500FT 5,000FT 4,001FT

Figure 6-3. Display of GROUP Named RUNWAY (Part 2 of 5)

20/AUG/85

PAGE 3 OF 11

RUNWAY INFORMATION

FLORIDA	NAME (CONTINUED)	SURFACE	CONDITION	LENGTH
	DAYTONA BEACH REGIONAL	ASPHALT	GOOD	7,500FT
	DELAND MUNI SIDNEY H TAYLOR	ASPHALT	FAIR	6,000FT
	DESTIN FT WALTON BEACH	ASPHALT	GOOD	5,000FT
	EGLIN AF AUX NR 3 DUKE	ASPHALT	GOOD	8,000FT
	EGLIN AFB	ASPHALT	GOOD	12,000FT
	FORT LAUDERDALE EXECUTIVE	ASPHALT	GOOD	6,000FT
	FT LAUDERDALE HOLLYWOOD INT	ASPHALT	GOOD	8,040FT
	GAINESVILLE REGIONAL	ASPHALT	GOOD	6,500FT
	HOMESTEAD AFB	CONCRETE	GOOD	11,200FT
	HURLBURT FIELD	ASPHALT	GOOD	9,600FT
	JACKSONVILLE INTERNATIONAL	CONCRETE	GOOD	8,000FT
	JACKSONVILLE NAS TOWERS FLD	ASPHALT	GOOD	8,000FT
	KEY WEST INTERNATIONAL	ASPHALT	GOOD	4,800FT
	KEY WEST NAS	ASPHALT	GOOD	10,000FT
	LAKE CITY MUNICIPAL	ASPHALT	GOOD	6,835FT
	LAKELAND MUNICIPAL	ASPHALT	GOOD	6,000FT
	MACDILL AFB	ASPHALT	GOOD	11,420FT

Figure 6-3. Display of GROUP Named RUNWAY (Part 3 of 5)

RUNWAY INFORMATION

FLORIDA	NAME	SURFACE	CONDITION	LENGTH
(CONTINUED)				
	MARIANNA MUNICIPAL	ASPHALT	GOOD	4,997FT
	MAYPORT NS	ASPHALT	GOOD	8,000FT
	MELBOURNE REGIONAL	ASPHALT	GOOD	9,481FT
	MIAMI INTERNATIONAL	ASPHALT	GOOD	13,000FT
	NAPLES MUNICIPAL	ASPHALT	FAIR	5,000FT
	NORTH PERRY	ASPHALT	GOOD	3,065FT
	OCALA MUNICIPAL	ASPHALT	GOOD	5,007FT
	OPA LOCKA	ASPHALT	GOOD	8,000FT
	ORLANDO EXECUTIVE	ASPHALT	GOOD	5,938FT
	ORLANDO INTERNATIONAL	CONCRETE	GOOD	12,004FT
	ORLANDO BEACH MUNICIPAL	ASPHALT	POOR	4,000FT
	PAGE FIELD	ASPHALT	GOOD	6,401FT
	PALM BEACH COUNTY PARK	ASPHALT	POOR	3,550FT
	PALM BEACH INTERNATIONAL	ASPHALT	GOOD	7,905FT
	PALESTINE CITY BAY COUNTY	ASPHALT	GOOD	6,004FT
	PATRICK AFB	ASPHALT	GOOD	9,022FT
	PENSACOLA NAS FORREST SHERM	ASPHALT	GOOD	8,000FT
	PENSACOLA REGIONAL	ASPHALT	GOOD	7,002FT
	PERRY FOLEY	ASPHALT	GOOD	5,013FT
	PETER O KNIGHT	ASPHALT	FAIR	3,400FT
	POMPANO BEACH AIRPARK	ASPHALT	FAIR	4,420FT
	SANFORD	ASPHALT	GOOD	8,000FT

Figure 6-3. Display of GROUP Named RUNWAY (Part 4 of 5)

20/AUG/85

PAGE 5 OF 11

RUNWAY INFORMATION

	NAME	SURFACE	CONDITION	LENGTH
FLORIDA	(CONTINUED)			
	SARASOTA BRAULENTON	ASPHALT	GOOD	7,000FT
	ST AUGUSTINE	ASPHALT	GOOD	6,947FT
	ST PETERSBURG CLEARWATER III	ASPHALT	GOOD	7,953FT
	TALLAHASSEE MUNICIPAL	ASPHALT	GOOD	8,000FT
	TAMiami	ASPHALT	GOOD	5,000FT
	TAMPA INTERNATIONAL	CONCRETE	GOOD	10,950FT
	TITUSVILLE COCOA	ASPHALT	GOOD	6,001FT
	TYNDALL AFB	CONCRETE	GOOD	10,000FT
	GEORGIA	VERO BEACH MUNICIPAL	ASPHALT	FAIR
WHITING FIELD NAS NORTH		ASPHALT	GOOD	6,000FT
WHITING FIELD NAS SOUTH		ASPHALT	FAIR	6,000FT
WITHAM FIELD		ASPHALT	GOOD	5,000FT
ALBANY/DOUGHERTY CO		ASPHALT	GOOD	6,001FT
ATHENS MUNICIPAL		ASPHALT	GOOD	4,989FT
BACON COUNTY		ASPHALT	GOOD	5,000FT
BAXLEY MUNICIPAL		ASPHALT	GOOD	3,800FT
BUSH FIELD		ASPHALT	GOOD	8,001FT
CAIRO GRADY COUNTY		ASPHALT	GOOD	3,000FT
CALLAWAY GARDENS HARRIS CO		ASPHALT	GOOD	5,000FT

Figure 6-3. Display of GROUP Named RUNWAY (Part 5 of 5)

ALL PAGES/ALL PARTS[DOWN]	displays all PAGEs of all PARTs within a single GROUP. The optional DOWN allows the display format to PAGE through each PART before beginning the next PART. The default is to PAGE across PARTs (with no PAGE incrementing) in a wrap-around format.
/ALL PARTS (/AP)	permits the user to view all PARTs within the current GROUP (PAGE number remaining the same).
<CR>	during interactive viewing, this command will cause a PAGE advance (providing one exists). during non-interactive viewing, the <CR> will cause the next <display command> specified to be processed. If all commands have been processed, the GDR prompt will appear.
END	terminates interactive display processing.
FIRST PAGE (FP)	displays the first PAGE of the currently specified PART.
<group name>	specifies that the viewing window will be the designated GROUP (PAGE1/PART1 is not specified).
LAST PAGE (LP)	displays the last PAGE of the currently specified PART.
/LAST PART (/LP)	displays the same PAGE number of the last PART of the current GROUP.
NEXT PAGE (NP)	displays the next PAGE of the current PART (unless you are already on the last PAGE in which case you will receive an error).
/NEXT PART (/NP)	displays the same PAGE number of the next PART in the current GROUP (unless you are already on the last PART in which case you will receive an error message).
<nnn>	displays the PAGE specified in the same PART.
/ <nnn>	displays the same PAGE in the PART number specified.
PAGE(P) <nnn> [THRU <nnn>]	displays the PAGE specified in the PART you are currently in. The optional THRU allows the user to view requested PAGEs one after another.

/PART(/P) <nnn> [THRU <nnn>] displays the same PAGE in the PART specified. The optional THRU allows the user to view all requested PARTs of the currently specified PAGE.

PREVIOUS PAGE (PP) displays the PAGE immediately before the current PAGE (unless you are already on the first PAGE in which case you will receive an error).

/PREVIOUS PART (/PP) displays the same PAGE in the PART immediately prior to the PART you are currently viewing (unless you are currently in the first PART which will cause an error message).

SAME PAGE (SP) this optional command will keep the display on the same PAGE. It is also the default if no PAGE command is specified.

/SAME PART (/SP) this optional command will keep the display on the same PART. It is also the default if no PART command is specified.

As a further example of how display commands are processed by GDR, the table below illustrates how paging commands would be processed against a <group name> that has 4 PARTs. It shows the starting screen display, the user's input, and the final screen display following the command's execution:

<u>START POSITION</u>	<u>USER'S INPUT</u>	<u>FINAL POSITION</u>
PAGE2, PART3	NP	PAGE3, PART3
PAGE1, PART2	NP/NP	PAGE2, PART3
PAGE3, PART4	NP	PAGE4, PART4
PAGE3, PART1	/NP	PAGE3, PART2
PAGE1, PART1	/PP	ERROR MESSAGE
PAGE1, PART3	Page 3/PP	PAGE3, PART2
PAGE1, PART1	PP	ERROR MESSAGE
PAGE1, PART3	NP/SP	PAGE2, PART3
PAGE2, PART4	/AP	Shows PAGE2 of all PARTs (sequentially)
PAGE3, PART1	Page 1 THRU 3/Part 1	Shows PAGES 1 THRU 3 of PART1 (sequentially)

6.5 Directing Book Output to Printer. Through the DIRECT BOOK command, the user has the capability to instruct GDR to create printed copies of the contents of any BOOK locally or to send the BOOK to a remote host. The syntax for the DIRECT BOOK command is:

```
DIRECT BOOK <book name> ON      [ ONLINE
                                   ON [REMOTE] <device-id> ]
    [TO <host name>]
    [COPIES <nnn>] IDENT "<ident line>" MARK <classification>.
```

The destination for the printed copies is defined as either an online printer or a remote device. The default is ONLINE, which will cause the BOOK to be directed to an online printer. Specifying ON, followed by a two character printer ID, will cause the BOOK to be directed to the specified remote line printer. The TO <host name> clause may be used to send the BOOK to another WWMCCS site. The copies clause may be used to specify the number of copies desired. The default is one copy. The IDENT clause must be specified. It contains the accounting information required by a batch job. This information must be enclosed in quotes. The MARK clause also must be specified. It denotes the classification of the printed copies. The classification caveat must be one of the following:

TZZ	Top Secret
SZZ	Secret
CZZ	Confidential
UZZ	Unclassified
ZZZ	No Classification

Upon acceptance of the DIRECT BOOK command, the system will respond with a SNUMB number which will serve to identify your printed output and must be used when picking your job up. For example, if a user entered:

DIRECT BOOK BKAIRFLD ONLINE IDENT "JNGG" MARK UZZ.

The system response would be something like:

SNUMB 2840T

APPENDIX A
GIPSY EXECUTION SEQUENCE

THIS PAGE INTENTIONALLY LEFT BLANK

GIPSY is composed of several independently executable load modules (H* files) whose execution sequence is dynamically determined based upon user input specifications and a predefined execution sequence initially loaded from GIPSY's cue library. The execution sequence is reloaded from the DAFC after each module is executed.

GIPSY is properly a TSS subsystem. It may be invoked from the terminal keyboard by responding "GIPSY" to the Time Sharing System (TSS) standard input request prompt or it may be invoked by a user program via DRL CALLSS. The FORTRAN callable subroutine CALLSS accomplishes this for high order languages (subroutine CALLSS is discussed in the Honeywell FORTRAN Subroutine Libraries Manual (order number DD20)). The FORTRAN calling sequence for invoking GIPSY from a user program is

```
CALL CALLSS ("GIPSY  command line syntax options  #")
```

The argument on this call must be in ASCII.

However GIPSY is activated, the first four characters are used to identify the command library kernel GIPS which is executed from CMDLIB/GIPS. This kernel is hard-coded to invoke the GIPSY executive controller from the H* file defined by the catalog file descriptor (cfd) LIBRARY/GIPSY/LOAD/..GIPSY. This will be overridden if a file named ..GIPSY is in the Available File Table (AFT) at the time of initiation. Control is transferred to the executive controller via DRL RESTOR. Prior to this transfer of control, the entire command line is retained and passed to the executive controller for syntactic and semantic processing.

The executive controller (..GIPSY) attaches the cue library (hard-coded as LIBRARY/GIPSY/LIBS/..CUELIB). The catalog file string for the environment and terminal definition (..ENVIRO) file and the execution sequence of all GIPSY modules are extracted from the cue library. The terminal is associated with the terminal definition extracted from ..ENVIRO based on the station code obtained from TSS. The command line statements are then processed. All command line substitutions are applied to the execution sequence. The execution sequence and all current semantics are written to the DAFC. The executive controller then copies the status of all GIPSY internal parameters to the DAFC and transfers control to the first module in the execution sequence. If any module name is null, ..GIPSY automatically proceeds to the next module in the sequence. If the executed module completes its task successfully, it updates the DAFC to reflect its successful execution and to reflect all alteration to global internal GIPSY parameters. If any module fails to inform the executive controller to continue, the GIPSY session is immediately terminated.

When the next module is placed into execution, it retrieves all relevant communications (from the previous executed module) from the DAFC. The execution sequence is also restored from the DAFC in order to pick up any dynamic modification to the execution sequence that may have been applied by

the executing module.

The cue library (LIBRARY/GIPSY/LIBS/..CUELIB) contains the catalog-file descriptor of all permanent files used by GIPSY (except of course the executive controller and the cue library).

The execution sequence for GIPSY is defined as a linked list starting at cue library message number 1000. Each message has a module identifier keyword followed by the catalog-file string of the module to be executed. The module identifier keyword is defined by virtue of its existence in the execution sequence. Thus, as many modules and keywords as desired may be created and added to the execution sequence linked list causing them to be automatically defined and executed as part of GIPSY.

The default keywords and associated modules are:

```
PRE-MOD
PRE-SYNTAX
SYNTAX LIBRARY/GIPSY/LOAD/..SYNTAX
POST-SYNTAX
PRE-DATSEL
DATSEL LIBRARY/GIPSY/LOAD/..DATSEL
POST-DATSEL
PRE-MTXGEN
MTXGEN LIBRARY/GIPSY/LOAD/..MTXGEN
POST-MTXGEN
PRE-DISPLA
DISPLA LIBRARY/GIPSY/LOAD/..DISPLA
POST-DISPLA
PRE-GEOMOD
GEOMOD LIBRARY/GIPSY/LOAD/..GEOMOD
POST-GEOMOD
PRE-GDRMOD
GDRMOD LIBRARY/ GIPSY/LOAD/..GDRMOD
POST-GDRMOD
```

Keywords which do not have a corresponding module name (catalog file descriptor) are considered to be null.

APPENDIX B
SYSTEM SUPPLIED SUBROUTINES

THIS PAGE INTENTIONALLY LEFT BLANK

GIPSY currently has a set of 32 system-supplied subroutines available to the user. They are as follows:

- \$BODY-LINE
- \$BODY-PART
- \$BUILD-PROCESS
- \$BREAK-STATUS
- \$COMPARE-STRING
- \$DATE
- \$END-INPUT-PROCESSING
- \$END-PROCESS
- \$EXIT
- \$FILL-STRING
- \$INITIAL-PROCESS
- \$ISPPTR
- \$LINES-LEFT
- \$MOVE-STRING
- \$MOVE-TEXT
- \$OUTPUT
- \$PAGE-NUMBER
- \$PICTURE
- \$PRINT-DATE
- \$PRINT-TIME
- \$RECORD-EXCLUDE
- \$RECORD-HOLD
- \$RECORD-INCLUDE
- \$RECORD-SET
- \$RECORD-READ
- \$SPLIT-CATALOG
- \$STRING-LENGTH
- \$TERMINALID
- \$TERMINAL-TYPE
- \$TIME
- \$TOTAL-PAGES
- \$USERID

Each of these subroutines is described in detail on the following pages.

In general, subroutines are executable statements that are invoked in field table structures, the QUALIFY command, and in the RETRIEVE command. Subroutines are denoted by a dollar sign. Not all subroutines have universal applicability throughout the system. Specifics are noted in the following paragraphs.

Some subroutines return values and are subsequently assigned in data moves (i.e., TODAY-DATE = \$DATE). Other subroutines do not return values but cause a specific action to happen (i.e., \$EXIT IF COUNT GT 100 will terminate field table processing when COUNT is greater than 100).

\$BODY-LINE

Subroutine \$BODY-LINE will return the number of the current body line count of a page being processed within the record output table. The syntax for this statement is:

```
$BODY LINE (<part number>) [<conditional expression>] .
```

where <part number> is the specific part for which you want the body line count.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will print a dividing line separating the header from the body of the report. The statement will be executed when the body line count for part 1 is equal to line 1:

```
DIVIDER = "-----".  
LINE DIVIDER IF $BODY-LINE(1) EQ 1.
```

\$BODY-PART

Subroutine \$BODY-PART will return the part number currently being processed in the record output table. The syntax for this statement is:

```
$BODY-PART [<conditional expression> ] .
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example illustrates this subroutine in a conditional expression:

```
GROUP INDEX = 1, WATERTYPES = 2:3:4:5:6:7.  
HEADER TABLE.  
  LINE "WATERWAYS OF THE WORLD"  
    CENTERED IF $BODY-PART BT 2/7.
```

The literal "WATERWAYS OF THE WORLD" will be centered in the heading whenever the current body part is part of the group called WATERTYPES (i.e., between 2 and 7).

\$BUILD-PROCESS

Subroutine \$BUILD-PROCESS accepts commands within a field table that are executed outside of field table structures. The \$BUILD-PROCESS subroutine is part of the build process block structure. The build process block structure consists of an opening statement, \$INITIAL-PROCESS, one or more

\$BUILD-PROCESS, and the closing statement (\$END-PROCESS). When the highest level of field table has been executed, the \$BUILD-PROCESS commands that were accepted in the field table are executed. The syntax for this statement is:

```
$BUILD-PROCESS      { "literal"      }    [<conditional expression> ] .  
                    { <variable>    }
```

where "literal" is a GIPSY command that is processed outside of a field table or the variable contains the GIPSY command.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, a book will be built based on a previously set flag. Note that the statement "TRANSFER TO GDR." was assigned to a variable and the variable is part of the BUILD-PROCESS:

```
FIELD TABLE.  
  TRA-CMD = "TRANSFER TO GDR."  
  $INITIAL-PROCESS.  
  $BUILD-PROCESS (TRA-CMD)  
  $BUILD-PROCESS ("BUILD OUTPUT WATER TO X.")  
    IF WTRFLAG.  
  $BUILD-PROCESS ("BUILD OUTPUT CITIES TO X.")  
    IF CTYFLAG.  
  $END-PROCESS.  
END.
```

\$BREAK-STATUS

Subroutine \$BREAK-STATUS returns a code indicating that a break has been entered on a Visual Information Projection (VIP) terminal. The syntax for this statement is:

```
$BREAK STATUS.
```

In the following example, the message "BREAK ACKNOWLEDGED" will be written to the screen (by a user-supplied subroutine) when a break is entered:

```
FIELD TABLE WHEN $BREAK-STATUS.  
  $DISPLAY ("BREAK ACKNOWLEDGED").  
  $DISPLAY-OUT.  
END.
```

\$COMPARE-STRING

Subroutine \$COMPARE-STRING compares a specified set of characters of one string with a specified set of characters of another, setting one of two logical variables to "true" based on the outcome of the comparison.

The syntax for this statement is:

```
$COMPARE-STRING    (<variable1>, <start1>, <length1>,  
                    <variable2>, <start2>, <length2>,  
                    <logical1>, <logical2>)  
                    [<conditional expression>] .
```

where <variable1> contains the character string to be compared.

where <start1> is the starting position within <variable1>. This may be a variable or a numeric literal.

where <length1> is the number of characters of <variable1> to compare. This may be a variable or a numeric literal.

where <variable2> contains the string to compare against <variable1>. This may also be a literal.

where <start2> is the starting position within <variable2>. This may be a variable or a numeric literal.

where <length2> is the number of characters of <variable2> to use in the comparison. This may be a variable or a numeric literal.

where <logical1> is the logical variable that is set to "true" when the comparison is exactly equal.

where <logical2> is the logical variable that is set true when the comparison is not equal and the characters in the string of <variable2>, are alphabetically greater than the characters compared with in <variable1>.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, the first 4 characters of TEXT are compared with the 4 characters of the literal FOUR. When there is an exact match, the logical variable EQUAL will be set to "true". When there is no match, and the literal FOUR is alphabetically greater than the contents of the data record, then the logical variable GT will be set to "true":

```
(Data File)  
ONE  
TWO  
THREE  
FOUR  
ELEVEN  
TWELVE  
THIRTEEN
```

FOURTEEN
FIFTEEN

```
BUILD QDT. ADD.  
  TEXT 1 A8.  
  EQUAL * L.  
  GT    * L.  
END.  
OUTPUT TABLE NAMED TEST.  
BODY TABLE.  
  $COMPARE-STRING (TEXT,1,4,"FOUR",1,4,EQUAL,GT).  
  LINE TEXT CENTERED IF EQUAL.  
  LINE TEXT RIGHT JUSTIFIED IF GT.  
END BODY.  
END OUTPUT.  
BUILD OUTPUT TEST TO XYZ.
```

A display of book XY7 would result in the following output:

FOUR	ELEVEN
FOURTEEN	FIFTEEN

GT was set true for only ELEVEN and FIFTEEN because the literal FOUR was alphabetically greater than either ELEV or FIFT.

\$DATE

Subroutine \$DATE accesses the system date kept internally in the mainframe and returns it to the user in the format MMDDYY. The result is placed in a previously defined field. This subroutine is not valid in the record output table. The syntax for this statement is:

\$DATE [<conditional expression>] .

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.5 for an explanation of conditional expressions.

The statement:

RUN-DATE - \$DATE.

will assign the current date to the field RUN-DATE during execution of the field table in which the statement resides. The field RUN-DATE must be a 6 character field.

\$END-INPUT-PROCESSING

Subroutine \$END-INPUT-PROCESSING is used within a field table for data selection. This subroutine, used in conjunction with a "SET AUTO FILE INPUT OFF." command, indicates the end of data selection. The syntax for this statement is:

```
$END-INPUT-PROCESSING [<conditional expression>] .
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, data selection will be exited when LENGTH is greater than 5,000 ft:

```
SET AUTO FILE INPUT OFF.  
.  
.  
.  
FIELD TABLE.  
.  
.  
.  
$END-INPUT-PROCESSING IF LENGTH GT 5000.  
END.
```

\$END-PROCESS

Subroutine \$END-PROCESS defines the closing statement of the build process block structure. The build process block structure consists of an opening statement, \$INITIAL-PROCESS, one or more \$BUILD-PROCESS statements and the \$END-PROCESS statement. The syntax for this statement is:

```
$END-PROCESS.
```

In the following example, the statement "TRANSFER TO GDR." will be executed after the field table containing the build process structure is executed and control is returned to GIPSY's command level:

```
FIELD TABLE.  
.  
.  
.  
$INITIAL-PROCESS.  
$BUILD-PROCESS ("TRANSFER TO GDR.").  
$END-PROCESS.  
END.
```

\$EXIT

Subroutine \$EXIT is used to suspend normal field table processing. When this subroutine is invoked with a nested field table structure, all levels of field tables are exited. Processing is returned to the GIPSY command immediately following the END statement of the highest level field table structure. The syntax for this statement is:

```
$EXIT [<conditional expression>] .
```

where <conditional expression> require a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions:

```
FIELD TABLE.  
  LENGTH = LEN:"FT".  
  .  
  .  
  .  
  FIELD TABLE.  
    $EXIT IF BAD-RECORD.  
    ADD 1 TO COUNT.  
    .  
    .  
    .  
  END.  
END.
```

The execution of these fields will cause the literal "FT" to be concatenated to the field LEN and will add 1 to the field "COUNT" if the logic variable BAD-RECORD is false.

\$FILL-STRING

Subroutine \$FILL-STRING uses the contents of a variable or a literal to completely fill all the positions in a receiving variable. The syntax for this statement is:

```
$FILL-STRING { <variable> } [<conditional expression>] .  
              "literal"
```

where <variable> or "literal" contains the string to be used for filling the receiving variable.

where <conditional expression> requires a set of conditions to be satisfied before subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will use the contents of variable STAR to fill the 20 character field ALPHA when PAX equals zero:

ALPHA = \$FILL-STRING (STAR) IF PAX EQ 0.

\$INITIAL-PROCESS

Subroutine \$INITIAL-PROCESS defines the opening statement of the build process block structure. The build process block structure consists of this opening statement, one or more \$BUILD-PROCESS statements and the closing statement, \$END-PROCESS. The syntax for this statement is:

\$INITIAL-PROCESS.

In the following example, the statement "SET SIZE LARGE." will be executed after the field table containing the build process structure is executed and control is returned to GIPSY's command level:

FIELD TABLE.

```
.  
. .  
. .  
$INITIAL-PROCESS.  
$BUILD-PROCESS ("SET SIZE LARGE.").  
$END-PROCESS.
```

END.

\$ISPPTR

Subroutine \$ISPPTR returns an Indexed Sequential Processor (ISP) record pointer value which is used to build an index file. This routine is only valid when the user's data file is an ISP file and when the subroutine is used prior to data selection. The subroutine value is assigned to previously defined field in the FDT structure. The syntax for this statement is:

\$ISPPTR.

In the following example, the field POINTER-VAL will contain a pointer to the beginning of each STATE-CODE after the field table is processed against the ISP data file:

```
BUILD FDT.  
  ADD.  
    POINTER-VAL * A6.  
END.  
FIELD TABLE WHEN STATE-CODE CHANGES.  
  POINTER-VAL = $ISPPTR.  
END.
```

\$LINES-LEFT

Subroutine \$LINES-LEFT will return the number of body lines left on the current page being processed within the record output table. The syntax for this

statement is:

```
$LINES-LEFT (<part number>) [<conditional expression>] .
```

where <part number> is the specific part for which the body line count of lines left on the page is being calculated.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example illustrates this subroutine used within a conditional expression to determine when the field table is to be executed:

```
FIELD TABLE WHEN $LINES-LEFT(1) EQ 0 AND NOT COUNTRY-CODE CHANGES.  
  SPACE 1.  
  LINE COUNTRY-NAME: "(CONTINUED)" COL 10.  
  SPACE 1.  
END.
```

Execution of the field table will cause the name of the country to be printed at the top of the next page provided there are no remaining lines on the current page and the country being processed does not change.

\$MOVE-STRING

Subroutine \$MOVE-STRING allows the moving of partial fields from one variable to another, allowing the precise placement of data within the destination variable. The syntax for this statement is:

```
$MOVE-STRING (<variable1>, <start1>, <chars1>,  
              <variable2>, <start2>, <chars2>)  
              [<conditional expression>] .
```

where <variable1> is the receiving field and must be defined as type alphanumeric.

where <start1> is the starting position within the receiving field to begin placing data. This may be a numeric literal or a variable field.

where <chars1> is the number of characters of data being moved. This may be a numeric literal or a variable field.

where <variable2> is the source field from which to extract the data and which must be defined as type alphanumeric.

where <start2> is the starting position within the source field from which to begin extracting data. This may be a numeric literal or a variable field.

where <chars2> is the number of characters being moved from the source field. This may be a numeric literal or a variable field.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, assume CLIST is defined as A80 and CHOLD is defined as A10. The first 2 characters of CHOLD will be moved into positions 5 and 6 of CLIST:

```
$MOVE-STRING (CLIST, 5, 2, CHOLD, 1, 2).
```

In the next example, ALPHA is defined as A40 and STAR as A1 containing a single asterisk. As a result of the execution of this subroutine, ALPHA will contain 1 asterisk in position 1 followed by 39 blanks:

```
$MOVE-STRING (ALPHA, 1, 40, STAR, 1, 40).
```

If no blanks are desired, chars1 and chars2 should be equal:

```
$MOVE-STRING (ALPHA, 1, 40, STAR, 1, 40).
```

After execution, ALPHA would contain 40 asterisks.

\$MOVE-TEXT

Subroutine \$MOVE-TEXT allows the positioning a variable string or literal within another variable string either centered, right-justified, or left-justified. The syntax for this statement is:

```
$MOVE-TEXT { <variable name> } , <pos nbr> [<conditional expression>] .  
            " literal"
```

where <variable name> is the field to be moved.

where <pos nbr> is the desired position number; valid numbers are "1" for centered, "2" for left-justified, and "3" for right-justified.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example centers the contents of the 6 character field COLOR within the 20 character field ALPHA:

```
ALPHA - $MOVE-TEXT (COLOR, 1).
```

\$OUTPUT

Subroutine \$OUTPUT is used within a field table for data selection. This subroutine writes the record being processed to the QDF. A \$OUTPUT subroutine will override a "SET AUTO QDF OUTPUT OFF" command. If a record output table is specified, the records will be written to the QDF in the record output table format. The syntax for this statement is:

```
$OUTPUT [<conditional expression>] .
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, the record will be written to the QDF when TYPE is equal to "A1" in the format "TYPE RECORD", type:

```
SET AUTO QDF OUTPUT OFF.  
FIELD TABLE.  
    $OUTPUT IF TYPE EQ "A1".  
END.  
RECORD OUTPUT TABLE.  
    FIELDS "TYPE RECORD", TYPE.  
END.
```

\$PAGE-NUMBER

Subroutine \$PAGE-NUMBER will output the current page number of the specified part within the record output table. The syntax for this statement is:

```
$PAGE-NUMBER (<part number>) [<conditional expression>].
```

where <part number> is the specific part for which the page number is to be output.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example the page number of the current value of XPART will be placed in variable PPAGE for later reference. This is useful when retaining page numbers for output as an index to the report:

```
PPAGE = $PAGE-NUMBER (XPART).
```

\$PICTURE

Subroutine \$PICTURE converts a binary integer field or integer field into its integer equivalent and formats the resulting number with a COBOL-like edit mask. The syntax for this statement is:

\$PICTURE (<variable name>, "<picture mask>") [<conditional expression>] .

where <variable name> is any previously defined field.

where <picture mask> is the COBOL-like edit mask (any combination of Z's, 9's, dollar signs, plus signs, minus signs, commas, or decimal points).

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will convert the binary integer in STATNUM, zero-suppress the result, and place it in OUTNUM. OUTNUM may be alphanumeric, integer, or binary integer and can be printed as a decimal number. Note that a binary integer is a 6 digit field:

```
OUTNUM = $PICTURE (STATNUM,"ZZZZZ9").
```

\$PRINT-DATE

Subroutine \$PRINT-DATE will output the date the report is generated in the record output table as an output element to a specified location. The syntax for this statement is:

```
$PRINT-DATE [<conditional expression>].
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, the current date will be left-justified in the heading for PART 1. The format for the date is dd/mm/yy (i.e., 11/MAY/86):

```
HEADER TABLE PART 1.  
  LINE $PRINT-DATE LEFT JUSTIFIED.
```

\$PRINT-TIME

Subroutine \$PRINT-TIME will output the current time as an output element within the record output table to a specified location. The syntax for this statement is:

```
$PRINT-TIME [<conditional expression>].
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will left justify the current time in the heading for PART 3. The format for the time is the standard military format (HH:MM):

HEADER TABLE PART 3.

LINE \$PRINT-TIME LEFT JUSTIFIED.

\$RECORD-EXCLUDE

Subroutine \$RECORD-EXCLUDE will exclude the current record from the QDF process regardless of the QUALIFY in effect. This routine is used within a field table QUALIFY. The syntax for this statement is:

\$RECORD-EXCLUDE [<conditional expression>].

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example illustrates this subroutine in a conditional expression.

Records will be excluded when QTYPE (which is a QUALIFY field) is equal to BAY:

```
FIELD TABLE QUALIFY.  
  QTYPE = TYPE.  
  $RECORD-EXCLUDE IF Q TYPE EQ BAY.  
END.
```

\$RECORD-HOLD

Subroutine \$RECORD-HOLD will hold the current record intact for the next qualified record request during QDF processing. The record that is held is evaluated against the same criteria as the next record that is processed. This subroutine is very useful when processing hierarchical QDF structures. It is used within a field table for QUALIFY. The syntax for this statement is:

\$RECORD-HOLD [<conditional expression>].

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, the record will be held for comparison with the next record's criteria when the field RECORD-NBR is not equal to the contents of the field NFMS:

```
FIELD TABLE QUALIFY  
  ADD 1 TO RECORD-NBR.  
  $RECORD-HOLD IF RECORD-NBR NE &NFMS.  
END.
```

\$RECORD-INCLUDE

Subroutine \$RECORD-INCLUDE will include the current record from the QDF process regardless of the QUALIFY statement. This routine is used within a field table for QUALIFY. The syntax for the statement is:

```
$RECORD-INCLUDE [<conditional expression>].
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, the subroutine will cause all records to be included when QCOUNTRY is equal to USA:

```
FIELD TABLE QUALIFY.  
  QCOUNTRY = COUNTRY.  
  $RECORD-INCLUDE IF QCOUNTRY EQ USA.  
END.
```

\$RECORD-SET

Subroutine \$RECORD-SET is used in conjunction with subroutines \$RECORD-INCLUDE and \$RECORD-EXCLUDE. Subroutine \$RECORD-SET will include or exclude the current record from the QDF based on the condition of the previous record. If the previous record qualified, the current record qualifies; if the previous record did not qualify, the current record does not qualify as well. The field table QUALIFY is activated by the QUALIFY command. The syntax for this statement is:

```
$RECORD-SET [<conditional expression>] .
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will process a sequence of numbers resulting in all numbers between 1 and <n> being processed except 03, 04, 07, 08:

```
FIELD TABLE QUALIFY.  
  QNUM = NUM.  
  $RECORD-SET.  
  $RECORD-INCLUDE IF QNUM EQ 01, 05, 09.  
  $RECORD-EXCLUDE IF QNUM EQ 03, 07.  
END.
```

\$RECORDS-READ

Subroutine \$RECORDS-READ will return the number of records read during data selection. The syntax for this statement is:

\$RECORDS-READ [<conditional expression>].

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, variable NUMREC provides a counter of the number of records read. Processing of records will be halted when NUMREC is greater than 15:

```
FIELD TABLE.  
  NUMREC = $RECORDS-READ.  
  $END-INPUT-PROCESSING IF NUMREC GT 15.  
END.
```

\$SPLIT-CATALOG

Subroutine \$SPLIT-CATALOG expands each entry in a given catalog file string to a full 12 characters, padding shorter entries with blanks and using the slash as a delimiter. The result is assigned to a receiving variable, which can then be used to check for valid catalog names or files. The syntax for this statement is:

```
$SPLIT-CATALOG    { variable } [<conditional expression>] .  
                  { "literal" }
```

where <variable> or "literal" contains the catalog file string to be expanded.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example uses a data file of catalog file strings and \$SPLIT-CATALOG to decide which field table to execute. In this case, when the catalog file string for FILE1 is processed, GIPSY is exited:

Data file:

```
837IDPX0/GIPSY/DATA/FILE09  
837IDPX0/GIPSY/DATA/FILE22  
837IDPX0/GIPSY/DATA/FILE1
```

```
BUILD FDT.  
  ADD.  
  CAT-STRING 1 A30.  
  CAT-CHECK * A60.  
END.
```

```
CAT-CHECK = $SPLIT-CATALOG (CAT-STRING).
```

```

FIELD TABLE WHEN CAT-CHECK (37/41) EQ FILE1
  $INITIAL-PROCESS
  $BUILD-PROCESS("//NOTE GIPSY TERMINATING").
  $BUILD-PROCESS("DONE. ").
  $END-PROCESS.

```

END.

\$STRING-LENGTH

Subroutine \$STRING-LENGTH returns the number of characters in a given variable string. The syntax for this statement is:

```
$STRING-LENGTH (<variable name>) [<conditional expression>].
```

where <variable name> is any 12 character field name defined in the FDT or QDT.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will find the length of the variable INPUT-STR and place it in the variable STR-LEN:

```
STR-LEN = $STRING-LENGTH (INPUT-STR).
```

\$TERMINALID

Subroutine \$TERMINALID returns the 2 character code for the terminal currently logged on. The syntax for this statement is:

```
$TERMINALID [<conditional expression>].
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example controls the processing of future statements by determining if the PCS is being executed in batch or TSS:

```

BUILD FDT.
ADD TO GLOBAL.
  TERMINALID * A2.
  BATCH      * L.
END.
FIELD TABLE FOR INITIAL.
  TERMINALID = $TERMINALID.
  BATCH = TRUE IF TERMINALID EQ "***".
END.

```

\$TERMINAL-TYPE

Subroutine \$TERMINAL-TYPE returns the terminal type currently logged on during run time, and as defined to GIPSY in the ..ENVIRO file. The syntax for this statement is:

```
$TERMINAL-TYPE [<conditional expression>] .
```

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will print out a 6 digit field, "000009", for a Tektronix 4027 terminal:

```
LINE $TERMINAL-TYPE LEFT JUSTIFIED.
```

The next example will set the output of the report based on the type of terminal:

```
SET OUTPUT 80 BY 30 IF $TERMINAL-TYPE EQ "000009".
```

\$TIME

Subroutine \$TIME accesses the system time kept internally in the mainframe and returns it to the user in standard military time format (i.e., 1600 for 4:00 pm). The syntax for this statement is:

```
$TIME.
```

The statement:

```
RUN-TIME = $TIME.
```

would assign the current time to the field RUN-TIME during execution of the field table in which the statement resides. This subroutine is not valid in the record output table.

\$TOTAL-PAGES

Subroutine \$TOTAL-PAGES returns the total number of pages for the specified part of a report being processed within the record output table assuming that the "TOTAL PAGES." command has been specified. The syntax for this statement is:

```
$TOTAL-PAGES (<part number>) [<conditional expression>] .
```

where <part number> is the specific part for which a total page count is desired.

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

In the following example, "PAGE n of nn" will be left-justified in the heading for PART 2:

HEADER TABLE PART 2.

LINE "PAGE " : \$PAGE-NUMBER(2) PIC "Z,ZZ9": " OF " : \$TOTAL-PAGES(2)

PIC "ZZ,ZZ9" COL 65 LEFT JUSTIFIED TEXT COMPRESSED.

\$USERID

Subroutine \$USERID returns the userid of the person currently logged on to the system. The syntax for this statement is:

\$USERID [<conditional expression>].

where <conditional expression> requires a set of conditions to be satisfied before the subroutine is executed. See section 3.2.6 for an explanation of conditional expressions.

The following example will print out the userid of the person processing the report:

LINE \$USERID LEFT JUSTIFIED.

The next example will prohibit a user from executing the PCS unless their userid is valid:

FIELD TABLE WHEN \$USERID NE "DJ8XI344CB"
"DJ8XC344AP"
"DJ8XI344CI"

\$INITIAL-PROCESS.

\$BUILD-PROCESS ("DONE.").

\$END-PROCESS.

END.

APPENDIX C
FIELD TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. FIELD TABLES

The Field Table is a set of commands and conditions for execution of those commands that may be used to modify data fields, fill in new data fields, move data around, perform counts and calculation, execute user-written subroutines, and a number of other data manipulation services. For discussion purposes, consider the following scenario:

Each record in the user's data base consists of a one-character service code (A = Army, F = Air Force), a one-character equipment code (T = Tanks, P = Planes, R = Rifles), and the total number of pieces of that equipment for that service. Using the GIPSY commands previously discussed in this manual, the user can produce a report (sorted on service code) that looks like this:

SERVICE EQUIPMENT NBR-EQUIP

A	T	517
A	P	249
A	R	388
F	T	151
F	P	1275
F	R	755

The above report is only meaningful to someone who knows the data base and the translation of the codes. A more meaningful report would be the following:

SERVICE	EQUIPMENT	NBR-EQUIP
ARMY	TANKS	517
	PLANES	249
	RIFLES	388
AIR FORCE	TANKS	151
	PLANES	1275
	RIFLES	755

GIPSY's data manipulation capability gives the user the tools he/she needs to produce meaningful output. These tools are contained in a structure called the Field Table. The Field Table is the basis of all data manipulation throughout GIPSY.

All field manipulation performed in a Field Table can be described as a data assignment. That data assignment may be a simple movement of data from one field to another or a very complex mathematical or composite assignment which is conditionally applied.

C.1 Data Fields

The Field Table data manipulation capabilities can operate on data fields

which exist in the user's file and described in the basic file or it can operate on newly created data fields that are added to the data record at execution time. The FDT building capability is used to define both of these types of fields. The Field Table is used to load data into these fields based on data retrieval from the file, processing conditions, and literals. Most field table data manipulation occurs in the classic assignment statement form:

<receiving field> - <assignment definition>.

The <receiving field> is typically a new field that is created by extending the record buffer as discussed below. The receiving field may be a field from the basic file. However, this is not normally done since it would destroy the original data extracted from the user data base. The discussions in this appendix will assume use of the extended fields since use of the basic data field appears to have limited value. The <assignment definition> may be anything from a literal assignment to a complex conditional assignment within a loop-controlled processing structure.

C.1.1 Extended Field Definition. An extended field is a field that is not in the user's data base but a "work space" for GIPSY to use. As each data base record is read into GIPSY's record processing area (the "record buffer"), an "extended field buffer" is logically connected to that record buffer. This extended field buffer is a totally separate, self-contained entity that has no impact on the record buffer.

The size of the extended field buffer is determined by the size of all of the defined extended fields. Extended fields can't be used in a Field Table structure until they are defined.

Defining extended fields to GIPSY is accomplished in the "BUILD FDT" structure which is discussed in section 3.2.5.1. The syntax used in defining extended fields is the syntax used to define data base fields:

<field name> <starting position> <data type> [<data length>]

The <field name> of an extended field has the same restrictions as the name of a data base field: a 12-character limit with no imbedded blanks and no duplication of names. With extended fields, the name duplication restriction applies not only to other extended field names but to data base field names as well. In other words, all fields being defined to GIPSY must have unique names.

The <starting position> of an extended field is not known by the user. GIPSY assigns starting positions to extended fields based upon the length of the previously defined extended field. Even the length of some field types is not known by the user so he/she cannot consistently compute the <starting position> of extended fields. GIPSY, however, does know the length of the previous field. An asterisk ("*") is used in the <starting position> location of the command to tell GIPSY that the field being defined is to be an extended field and to assign a <starting position> to that field.

As with data base field definitions, each extended field can have subordinate fields associated with it. For instance, a DATE field of six characters may also be broken into DAY, MONTH, and YEAR fields. The user wants to be able to reference the entire DATE as well as the individual DAY, MONTH, and YEAR in the retrieval. The section of the "BUILD FDT" structure that defines this configuration is:

```
DATE      *      A6
DAY       DATE(1) A2
MONTH     DATE(3) A2
YEAR      DATE(5) A2
```

The DAY field starts in the first character of DATE, the MONTH field starts in the third character of DATE, and the YEAR field starts in the fifth character of DATE.

The <data type> and <data length> criteria used in defining extended fields are identical to those used for the definition of data base fields. The various data types are outlined below for ease of reference:

<u>GIPSY Code</u>	<u>Definition</u>	<u>Data Length Given</u>
A	Alphanumeric	YES
I	Integer	YES
F	Floating Point	YES
C	Coordinate	YES
L	Logic	NO
BI	Binary Integer	NO
BF	Binary Floating Point	NO

If each record of a user's data base is 30 characters in length and two extended fields for some data manipulation purpose are desired, the "BUILD FDT" structure might look like this:

```
BUILD FDT.
ADD.
  FIELD-A    1    A6.
  FIELD-B    7    A6.
  FIELD-C   13    I3.
  FIELD-D   16    C15.
  FIELD-E    *    A2.
  FIELD-F    *    A12.
END.
```

Because there is no actual connectivity between the record buffer and the extended buffer, file fields and extended fields can be intermixed when defining the FDT.

```

BUILD FDT.
ADD.
  FIELD-B      7      A6.
  FIELD-F      *      A12.
  FIELD-D      16     C15.
  FIELD-E      *      A2
  FIELD-C      13     I3.
  FIELD-A      1      A6.
END.

```

C.1.2 The Field Table Structure. The field table is a block structure. This block consists of a field table statement, one or more executable statements, and a terminating statement. The formal format of the block structure is:

```

FIELD TABLE [<type identifier>] [<execution control clauses>]
  executable statements
END [<end options>]

```

There are several types of field tables. Each type of field table will be discussed in section C.2.

It is important to note that the definition of a field table and its execution take place at two different times during a GIPSY run. This subparagraph concerns itself only with the definition of the overall "block" structure. Field table execution is discussed in the paragraphs that discuss specific field tables.

C.1.2.1 The Field Table Statement. The opening statement of a field table provides all of the controlling information for subsequent field table execution. The controlling information includes:

- a. Field Table type
- b. Field Table name
- c. When to execute the Field Table
- d. How many times to execute the Field Table

The specific syntax for the opening sentence of each type of field table is given in the paragraph that discusses that field table type. The general syntax of the opening sentence is:

```

FIELD TABLE [<type identifier>] [FOR <for clause>] [NAME <name>]
[WHEN <when clause>] [REPEAT <repeat clause>].

```

The <identifier> must be the first piece of any field table structure opening sentence. All of the other pieces (NAMED, FOR, WHEN, and REPEAT) are order-independent and optional. The following paragraph discusses each component part of the opening sentence.

The <type identifier> denotes the specific type of field table that is being built. It generally consists of the keywords FIELD TABLE followed by a keyword identifying the structure to which it applies. The particular <identifier> required for each specific field table type is defined in the paragraph that describes that field table type.

The <for clause> denotes which data record (first, last, intermediate) is to be used. The key word "FOR" must precede the <for clause>. The options available in the <for clause> and which data record each option specifies for use during execution of this field table structure are:

<u>For Clause</u>	<u>Data Record</u>
INITIAL	First record
FILE	Intermediate records
FINAL	Last record

The <name clause> is used to assign an understandable name to a field table. The naming requirements for each individual type of field table are discussed in section C.2. The name can be up to 36 characters in length. Imbedded blanks are not allowed in the name; a hyphen ("-") may be substituted for a blank in order to separate words in a name. If the user has a field table that builds a meaningful date from an integer data field, the name the "BUILD DATE FIELD" could not be used. However, the name "BUILD-DATE-FIELD" could be used. One very important note about the <name clause>. All field tables with the same name are combined into one structure upon definition.

Both the <when clause> and <repeat clause> use standard conditional expressions as defined earlier in this manual. Any embedded conditional expression must be terminated with a semicolon (;). The <when clause> is used to tell GIPSY when to execute the field table. Only data records which satisfy the <when clause> will be processed against the body of the field table.

The <repeat clause> is used to tell GIPSY how many times the field table is to be executed:

```
WHILE <conditional expression>
UNTIL <conditional expression>
FOR <n> TIMES [OR UNTIL <conditional expression>]
VARYING <varying parameters>
```

The parameter <n> in the FOR phrase is the number of times to repeat the field table. It may be a numeric constant, a mathematical expression (enclosed in parentheses), or a field name.

The structure of the <varying parameters> is <vary field> FROM <start> TO <end> [BY <increment>] [OR UNTIL <conditional expression>].

The <vary field> is the field which is to serve as a counter for comparison

against the <end> number for determining the execution termination. This field must be an extended field.

The <start> parameter is the initial value to which the <vary field> is set. It may be a numeric literal, a mathematical expression, or field name.

The <end> parameter is value against which <vary field> is compared. The field table structure will not execute once <vary field> is greater than <end>. It may be a numeric literal, a mathematical expression, or field name.

The <increment> parameter is the numeric value by which the <vary field> will be incremented after each execution of the field table body. If not specified, a value of one is assumed. The <increment> may be a numeric literal, a mathematical expression, or field name.

REPEAT WHILE will cause the field table to repeat only while the conditional expression is true. The truth of the conditional expression is tested prior to execution. Therefore, the minimum number of executions is zero.

REPEAT UNTIL will cause the field table to repeat until the conditional expression is true. The truth of the conditional expression is tested following execution. Therefore, the minimum number of executions is one.

REPEAT FOR will cause the field table to repeat the number of times specified. The current repeat count (held internally by GIPSY) is incremented and tested against the number of times specified following execution. Repeat execution will terminate as soon as the current count is greater than the specified number. Therefore, the minimum number of executions is the number of times specified, or zero if the specified number of times is equal to zero or it is a negative number (unless the optional OR UNTIL clause is also specified). If the OR UNTIL clause is present on a REPEAT FOR, the minimum number of executions will be one since the conditional expression of the OR UNTIL clause is tested after execution.

REPEAT VARYING, will cause the field table to repeat the number of times specified in the FROM/TO/BY span. The current repeat count (held by the <vary field> and beginning at <start>) is incremented and tested against the <end> count. Repeat execution will terminate as soon as the value in <vary field> is greater than the <end> count. The minimum number of executions is determined by the FROM/TO/BY span unless the optional OR UNTIL clause is also specified. If the OR UNTIL clause is present on a REPEAT VARYING, the minimum number of executions will be one since the conditional expression of the OR UNTIL clause is tested after execution. Upon termination, the <vary field> retains the last value assigned regardless of the reason execution was terminated.

Some examples of the field table statements and their meanings during execution are listed below:

FIELD TABLE.

This denotes that a field table is to be executed against every record in the data base.

FIELD TABLE WHEN SERVICE-CODE EQ ARMY.

This statement denotes that a field table is to be executed against each record of the data base only when the field "SERVICE-CODE" of the record is equal to "ARMY".

FIELD TABLE FOR INITIAL REPEAT VARYING COUNTER FROM 1 TO 10.

This statement denotes that a field table will be executed against the first record of the data base until the extended field "COUNTER" is greater or equal to ten. "COUNTER" will be incremented by one (the default) after every execution of the field table structure. Note that this also means that the field "COUNTER" will be equal to "10" after the final execution of field table structure.

FIELD TABLE WHEN SERVICE-CODE EQ A; REPEAT 5 TIMES.

This field table will be executed five times against each record in the data base whose field "SERVICE-CODE" is equal to "A". Note the semicolon (";") which separates the <repeat clause> and the <conditional expression> from the <when clause>.

FIELD TABLE REPEAT UNTIL SERVICE-CODE CHANGES; WHEN SERVICE-CODE EQ A.

This field table will be executed against each record in the data base whose field "SERVICE-CODE" is equal to "A" and will be repeated until the field "SERVICE-CODE" changes from "A" to something else. Note the semicolon (";") which separates the <repeat clause> and the <conditional expression> from the <when clause>.

FIELD TABLE REPEAT WHILE NAVY.

This field table will be executed against a record in the data base as long as the logic field "NAVY" is set to "TRUE". Once the logic field "NAVY" is set to "FALSE", this field table structure will stop executing when the end of the structure is reached. If the logic field "NAVY" is "FALSE" at the start, this structure will never execute.

FIELD TABLE FOR FINAL REPEAT FOR 10 TIMES OR UNTIL STOP-CODE.

This statement denotes that the field table will be executed only after the last record in the data base is read. It will stop executing when it has either executed 10 times or the logic field "STOP-CODE" is set to "TRUE".

C.1.2.2 The Executable Statement. There are seven types of executable statements which may be included in the field tables. The types are:

- a. Alphanumeric Move - to move alphanumeric data to another field
- b. Numeric Assignment - to assign the value of a number field to another field
- c. Mathematical Assignment - to capture the result of a computation
- d. User Subroutine - to perform some function not supported by GIPSY itself
- e. Field Routine - to execute a predefined set of data manipulations
- f. Concatenated Move - to concatenate several pieces of data into a single field
- g. Nested Field Table - to execute a previously defined field table.

Each type is discussed below. The executable statements of a field table must be placed between the opening sentence and the terminating sentence at the field table.

The general form of an executable statement is

<receiving field> = <assign clause> [IF <conditional expression>].

The <receiving field> is a pre-defined field which is to receive the result of the <assign clause>. Data will be assigned to the <receiving field> according to the type that is assigned to that field. Data moved into an alphanumeric <extended field> will be left-justified and blank-filled. Data moved into an integer or real <extended field> will be right-justified and blank-filled.

The <assign clause> identifies results which are to be moved to the <receiving field>. All data manipulation occurs in the <move field> portion of the statement.

There are some exceptions to this general form. Those exceptions will be discussed with the detailed syntax.

The illustrations in the remaining discussion will be based on the following sample FDT:

```
BUILD FDT.  
ADD.  
  SERVICE-CODE      1 A1.  
  BASE-NAME         2 A13.  
  EQUIP-CODE        15 A1.  
  NBR-EQUIP         16 I4.
```

END

The extended fields (starting position "*") contain no data, as nothing has been moved into them yet.

```
<receiving field> = "<literal>" IF { <conditional expression>
                                     <fieldname> } .
```

SERVICE-NAME - SERVICE-CODE.

SERVICE-NAME - "ARMY".

C-11

Partial fields (as discussed in section 3.2.1), may be used anywhere within this statement. Partial field notation allows a user to reference only part of a field. Briefly, a field name followed by a partial field notation span references only those characters of the first included in that span. If the user wants to reference only the first three characters of the field "BASE-NAME" in a straight move, the following statement would do the job:



The above set of statements would build a field in BIG-LIT that contained the literal "BASE CODE FOR FULL BASE NAME FORT HUACHUCA".

```
<receiving field> - [<numeric literal>] IF { <conditional expression> }
                                     <numeric field> }
```

There are many uses for the numeric move. It can be used to initialize an <extended field> that will be used in another executable statement that resolves a mathematical equation or counts occurrences of data or accumulates numerical data. It can be used to build a numeric scaling factor.

Referring to the example data base record and FDT, the following executable statement will set <extended field> READY-SCALE to zero:

```
READY-SCALE = 0.
```

The execution of the move is controlled within a conditional expression. For example, the <extended field> READY-SCALE may be set to a 0-9 scaling factor based upon the percent of total equipment that is completely ready to use (the record's field READY-PANT) by:

```
READY-SCALE = 9 IF READY-PANT BT 91/100.  
READY-SCALE = 8 IF READY-PANT BT 81/90.  
READY-SCALE = 7 IF READY-PANT BT 71/80.  
READY-SCALE = 6 IF READY-PANT BT 61/70.  
READY-SCALE = 5 IF READY-PANT BT 51/60.  
READY-SCALE = 4 IF READY-PANT BT 41/50.  
READY-SCALE = 3 IF READY-PANT BT 31/40.  
READY-SCALE = 2 IF READY-PANT BT 21/30.  
READY-SCALE = 1 IF READY-PANT BT 11/20.  
READY-SCALE = 0 IF READY-PANT BT 0/10.
```

The above set of statements would set <extended field> READY-SCALE to 9 only if the data base record's field READY-PANT were between 91 and 100, READY-SCALE would be set to 8 only if the record's field READY-PANT were between 81 and 90, and so on. After execution of this set of statements, the user has built a field that can be referenced and which contains a scaling factor.

Another variation of the numeric move is an executable statement called a "numeric assignment". A numeric assignment is an English-like mathematic expression that can only perform a single-level computation. The syntax of the four different numeric assignments available is presented below:

```
ADD [VALUE] <value> TO <extended field>.
```

where <value> is the amount to be added to the <extended field>

```
SUBTRACT [VALUE] <value> FROM <extended field>.
```

where <value> is the amount to be subtracted from the <extended field>

```
MULTIPLY <extended field> BY <value>.
```

where <value> is the amount by which <extended field> is to be multiplied

```
DIVIDE <extended field> BY <value>.
```

where <value> is the amount by which <extended field> is to be divided

In all four numeric assignments, <value> can be either a numeric literal, a

mathematic expression (surrounded by parentheses), or another field. The key word "VALUE" is to be used when partial field notation could be confused with division. For instance, at the beginning, GIPSY could not tell the difference between these two statements:

ADD (1/3) = TOTAL-EQUIP.
ADD (1/3) TO TOTAL-EQUIP.

To keep from confusing GIPSY, the key word "VALUE" is used before the mathematical expression to tell GIPSY that this is what you are doing. This is only necessary in the ADD and SUBTRACT assignments when you have a field named ADD or SUBTRACT.

The following examples show the various configurations of the numeric assignment and describe the computation performed:

ADD 1 to COUNTER. This numeric assignment would increment the value in the field COUNTER by one (COUNTER = COUNTER + 1).

SUBTRACT 10 FROM COUNTER. This numeric assignment would decrement the value in the field COUNTER by 10 (COUNTER = COUNTER - 10).

MULTIPLY COUNTER BY (OFFSET / 2). This numeric assignment would multiply the value in the field COUNTER by the result of the value in the field OFFSET divided by two. The final result would then be placed in the field COUNTER (COUNTER = (COUNTER * (OFFSET / 2))).

DIVIDE COUNTER BY OFFSET. This numeric assignment would divide the value in the field COUNTER by the value in the field OFFSET with the final result being placed in the field COUNTER (COUNTER=COUNTER/OFFSET).

C.1.2.2.3 Mathematical Assignment. The mathematical assignment statement is in the form:

<receiving field> = (<arithmetic assignment>) [IF <conditional expression>].

This statement allows the receiving field to be set as a result of a mathematical computation from data in the file or data added by some other GIPSY function or prior field table statement. The arithmetic expression follows the standard GIPSY definition but includes extended fields as valid fields. Note that the arithmetic expressions must be completely enclosed in parentheses. For a discussion of standard GIPSY arithmetic expressions refer to section 3.2.9.

GIPSY uses the same mathematical resolution precedence algorithm that is standard with most computers, including the Honeywell 6000 series: parenthetical expression, multiplication and division followed by addition and subtraction. In other words, the expression (((Y * X) / Z + A) - 1) / 2 would first get the result of Y * X, divide that by Z, add A to that result, subtract 1, and then divide the result by 2. Note that GIPSY does not support

exponentiation.

Using the example FDT to build an <extended field> with a count of the number of pieces of equipment in tens. The statement would be:

EQUIP-TENS = (NBR-EQUIP / 10).

Considering the example data base record and FDT, with EQUIP-TENS defined as an integer field, integer-type movement is performed. This means that the result in the <extended field> EQUIP-TENS will be rounded down to the nearest 10. If the user wishes to have EQUIP-TENS forced up to the nearest 10, the following executable statement would do the job:

EQUIP-TENS = ((NBR-EQUIP + 9) / 10).

GIPSY does not round in the conversion of EQUIP-TENS to the specified integer format.

C.1.2.2.4 User Subroutine Move. The format of this executable statement is:

[<receiving field>] = \$[(<arguments>)] <user subroutine>
[IF <conditional expression>].

The user subroutine move allows the data to be placed in the specified field based upon execution of a user-written subroutine. A user subroutine is a standard, Honeywell-executable subroutine. An executable subroutine is always in library element (loadable) format. Loadable format means that all external references in a subroutine are resolved. Any user subroutine that is to be used in an executable statement must reside on a LIBRARY File and must have an entry on the ...ARG file. The LIBRARY file is a standard Honeywell Q* housing one or more library elements. The ...ARG file is a file used by GIPSY that defines the Q* elements and their arguments to GIPSY. GIPSY has a pre-defined LIBRARY file and ...ARG file available to the user. The user does not have to do anything special to access the GIPSY user subroutine LIBRARY File in order to execute a GIPSY user subroutine.

All user subroutines used in an executable statement must be preceded by a dollar sign ("\$"). For our discussion here, the user subroutine "TIME" (which is on the standard GIPSY LIBRARY File and, therefore, available to all users) will be used. A very brief description of "TIME" follows so the reader can fully understand the user subroutine move.

The user subroutine "TIME" simply accesses the system time kept internally in the Honeywell mainframe and returns it to the user in standard military time format (i.e., 1600 for 4:00 PM). The resultant time is passed to the user in the <extended field> half of the executable statement.

The statement:

RUN-TIME - \$TIME.

would assign the current time to the field RUN-TIME. During execution of this statement, GIPSY calls the user subroutine "TIME" and places the current Honeywell system time into <extended field> RUN-TIME.

A few user subroutines do not return anything to the user. For instance, the user subroutine "DISPLAY" which simply displays a message on the terminal returns nothing back to the user. It does, however, have an input argument: the message to be displayed. For those "non-returning" user subroutines, there cannot be an <extended field> half of the executable statement. The statement used to display the message "ALL IS WELL" on the terminal is

\$DISPLAY ("ALL IS WELL").

When this statement is executed, GIPSY calls the user subroutine DISPLAY which displays the message "ALL IS WELL". Note that the user subroutine arguments other than the <extended field> return argument are surrounded by parentheses.

User subroutines are extremely useful in many ways. They give the user control and flexibility by providing the ability to "customize" GIPSY.

C.1.2.2.5 Field Routine. A field routine is a set of previously defined field table executable statements that have been assigned a name by which they can be activated. Any field routine used in an executable statement must be defined before it can be referenced. Field routines are defined for a specific type and the field routine type must be a type that is valid for the field table executing it. Field routines can be defined as field table subroutines.

All field routines used in an executable statement must be preceded by double dollar signs ("\$\$"). Assume an imaginary user (who is utilizing the example data base record and FDT) has defined a field routine named "BUILD-NEW-TIME-FORMAT". Assume that this particular field routine takes a military-format time and converts it to a regular time format. For example, "1600" is converted to "4:00 PM". This field routine returns data (the newly formatted time) to the user when it is executed; the statement using this field routine must have an <extended field> half to house the new time. Obviously, this routine must have access to the military-format time in order to reformat it to a regular time. Assume that this field routine performs the executable statement "RUN-TIME - \$TIME." as one of its functions. The following statement would return the newly formatted time to the <extended field> NORMAL-TIME:

NORMAL-TIME - \$\$BUILD-NEW-TIME-FORMAT.

Taken a step further, the "BUILD-NEW-TIME-FORMAT" field routine does not have to return data to the user. It could simply move the newly formatted time field directly into the <extended field> NORMAL-TIME as part of its executable statement. In this case, the executable statement that would perform this

entire function would be:

`$$BUILD-NEW-TIME-FORMAT.`

C.1.2.2.6 Concatenated Move. This type of data manipulation allows multiple data elements to be put together and then referenced as one data base record <fields>, <extended fields>, and <literals> may be concatenated into a single field by the concatenated move. This statement has the format:

`<receiving fields> = <concatenation fields> [IF <conditional expression>].`

The concatenation fields may be two or more fields and/or literals connected by a concatenation operator.

The concatenation operator is the colon (:). That is, in order to concatenate multiple data elements together, they must be connected by a colon (":"). This type of move provides a very flexible and powerful data manipulation tool.

Referring once again to the example data base record and FDT, the data base record's field READY-PANT contains the percent of the specified equipment that is in "READY" status. In our example data base record, 92% of EQUIP-CODE "T" is "READY". In the previous sentence, the percent sign following "92" made it obvious that percentages were being discussed. The user may also wish to use a percent sign in his/her report. The following statement accomplishes this:

`PERCENT-LIT = READY-PANT:"%".`

After execution of this statement, <extended field> PERCENT-LIT contains the data "92%" followed by a blank (remember PERCENT-LIT is defined as an A4 field). If the user wishes to build an <extended field> that would fully explain what the number in the field READY-SCALE is, the following executable statement could be used provided the <extended field> READY-SCALE had already been built:

`SCALE-LIT = "READINESS SCALE FACTOR = ":READY-SCALE.`

After this statement is executed, the <extended field> SCALE-LIT contains the string "READINESS SCALE FACTOR = 9".

If the user wants to build a data element that leaves no question about the meaning of the contents of the data built in EQUIP-TENS, the following two statements would be necessary in order to build that data element:

`EQUIP-TENS = (NBR-EQUIP / 10).`

`TENS-LIT = "AMOUNT OF EQUIPMENT (IN TENS) = ":EQUIP-TENS.`

Using the example data base record, the <extended field> TENS-LIT contains the string "AMOUNT OF EQUIPMENT (IN TENS) = 51. Two statements were necessary since only fields and literals are available for use in the <move field> half

of a statement.

To build clear text around RUN-TIME, the following executable statements could be used:

```
RUN-TIME = $TIME.  
MIL-TIME-LIT = "RETRIEVAL RUN AT":RUN-TIME:" HOURS".
```

The system time (returned in military time format from the user subroutine "TIME") is returned from the user subroutine "TIME" into <extended field> RUN-TIME in the first statement. The second statement then plugs RUN-TIME into the middle of <extended field> MIL-TIME-LIT. After execution, MIL-TIME-LIT would contain something like "RETRIEVAL RUN AT 1600 HOURS".

The user could build a field containing the regularly formatted time field in a similar fashion:

```
$$BUILD-NEW-TIME-FORMAT.  
NORMAL-LIT = "NORMAL TIME = ":NORMAL-TIME.
```

Remembering that the field routine "BUILD-NEW-TIME-FORMAT" places a converted military-to-regular time in the <extended field> NORMAL-TIME, NORMAL-LIT would contain the string "NORMAL-TIME = 4:00 PM" after execution.

C.1.2.2.7 Nested Field Tables. The field table may contain imbedded field tables. There is no limit to the number of "nested" field tables a user can have in a main structure. The basic syntax of an opening sentence as an executable statement is as follows:

```
FIELD TABLE [WHEN <when clause>] [REPEAT <repeat clause>].
```

Everything mentioned about both the <when clause> and the <repeat clause> in section C.1.2.1 applies here. The <when clause> is a conditional expression; the <repeat clause> is any one of the four available repeats: WHILE, UNTIL, FOR, VARYING. Notice that the <for clause> and the <name> are not available when defining the nested structure; all nested field tables are assigned to those belonging to the main structure.

The following scenario presents an example that introduces the use of this executable statement. Assume that the record is:

```
ATFLGATNALVANYNHAKAZTXAROKVTHICANMNVIDILWYNEOHKY
```

The BUILD FDT to define the data base records is:

```
BUILD FDT.  
ADD.  
  SERVICE-CODE  1  A1.  
  EQUIP-CODE    2  A1.  
  STATES       3  A100.
```

END.

Operationally, the fields are:

<u>Field Name</u>	<u>Meaning of Data</u>
SERVICE-CODE	Field length of one, starting in position one, containing an alphanumeric code for service
EQUIP-CODE	Field length of one, starting in position two, containing an alphanumeric code for type of equipment
STATES	Field length of 100, starting in position three, that may or may not be completely filled, giving a list of two digit alphanumeric state codes having responsibility for a least one piece of the service's equipment.

The user wants to build a report that will show the full name of the service, the full name of the equipment, and a count of states with that equipment. Obviously, the user is going to need to perform some data manipulation and his/her "BUILD FDT" structure defining his extending fields looks like this:

```
BUILD FDT
ADD.
    SERVICE-NAME * A10.
    EQUIP-NAME   * A6.
    STATE-COUNT  * I2.
    STATE-HOLD   * A2.
    NBR-STATES   * I2.
END.
```

The user has built a user subroutine called "STATE-CODE-RETURN". This subroutine returns to the next state code in the list held in the field STATES. The argument passed this subroutine is the current pointer into STATES.

The following field table will build understandable output fields from the near-nonsense data record shown above (which was probably constructed for data base storage efficiency).

```
FIELD TABLE.
SERVICE-NAME = "ARMY" IF SERVICE-CODE EQ A.
SERVICE-NAME = "AIR FORCE" IF SERVICE-CODE EQ F.
SERVICE-NAME = "MARINE" IF SERVICE-CODE EQ M.
SERVICE-NAME = "NAVY" IF SERVICE-CODE EQ N.
//
EQUIP-NAME = "PLANES" IF EQUIP-CODE EQ P.
EQUIP-NAME = "RIFLES" IF EQUIP-CODE EQ R.
EQUIP-NAME = "SHIPS" IF EQUIP-CODE EQ S.
EQUIP-NAME = "TANKS" IF EQUIP-CODE EQ T.
```

```
//
FIELD TABLE REPEAT VARYING STATE-COUNT FROM 1 to 99 BY 2
                OR UNTIL STATE-HOLD EQ " ".
STATE-HOLD = $STATES (STATE-COUNT).
END.
//
NBR-STATES = ((STATE-HOLD - 1) / 2).
ADD 1 TO NBR-STATES IF STATE-HOLD NE " ".
END.
```

The comments and indentation do not affect execution. They are added for readability and comprehension.

Presented below is an English breakdown of the functional blocks presented in the example above:

Build the extended field SERVICE-NAME so that it contains a meaningful service name for every record in the data base.

Build the extended field EQUIP-NAME so that it contains a meaningful equipment type name for every record in the data base.

Build the extended field STATE-HOLD by executing the subroutine named "STATE-CODE-RETURN" repeating it until either the end of the STATES field is reached or there are no more state codes in the field. Since there are only 50 states, STATE-HOLD will be built a maximum of 50 times. The extended field STATE-COUNT will start at "1" and be incremented by "2" every time STATE-HOLD is built. When either the end of the STATES field is reached or there are no more state codes left in the field, STATE-COUNT will be equal to one character less than the total number of valid state code characters. (If only one valid state code is found, STATE-COUNT will be set to "1" even though each valid state code is two characters long; if five valid state codes are found, STATE-COUNT will be set to "9"; etc.)

Build the extended field NBR-STATES so that it contains the number of states that have responsibility for at least one piece of equipment specified in EQUIP-CODE for the service specified in SERVICE-COUNT that was computed in the previous functional block. Since STATE-COUNT is not quite equal to the total number of valid state code characters found in the STATES field, subtract "1" from it. And, since that result is still two times greater than the actual number of states having this equipment for this service, divide by "2". Add "1" to this result if there were 50 valid state codes on the record (this is indicated by STATE-HOLD NE "BLANK").

Without the nested field table the following statements would have to be used to replace the third functional block (the nested field table):

```

NBR-STATES - 0
STATE-END - FALSE.
STATE-HOLD - STATES (1/2).
STATE-END - TRUE IF STATE-HOLD NE " " AND NOT STATE-FND.
ADD 1 TO NBR-STATES IF STATE-FND.
STATE-HOLD - STATES (3/4) IF STATE-FND.
STATE-FND - TRUE IF STATE-HOLD NE " " AND NOT STATE-END.
AND 1 TO NBR-STATES IF STATE-FND.
STATE-HOLD - STATES (5/6) IF STATE-FND.
STATE-FND - TRUE IF STATE-HOLD NE " " AND NOT STATE-FND.
ADD 1 TO NBR-STATES IF STATE-FND.
.
.
.

```

The last three statements would be entered until the statement that builds STATE-HOLD reached STATES (99/100). That is an extensive amount of typing, if nothing else. Also, this functional block is exceedingly difficult to understand. There are more disadvantages: this structure actually takes longer to process and requires much more memory.

The ability to nest field tables is extremely helpful when a user must develop very complicated retrievals from very complicated or hierarchical data bases.

With these tools, the user can manipulate data from any data base and build retrievals that can be easily understood and maintained.

C.2 Field Table Types

There are four types of field tables within GIPSY. The types are determined by the purpose which the field table serves and is identified by the <type identifier> on the field table statement. In the general syntax definition:

```
FIELD TABLE [<type identifier>] [<execution control clauses>]
```

If no <type identifier> is specified, that field table will be used in the data retrieval process and will be applied to every record in the file subject to the <execution control clauses> and the imbedded conditional expression specified. Type identifiers are:

QDF	to build a field table which will be processed against all QDF records which meet the criteria defined on the QUALIFY statement (as discussed in sections 3.4.4 and 4.1.2).
QUALIFY	to build a field table that will be processed against all records in the QDF before the QUALIFY statement is executed.
CALL	to build a field table which may be invoked for execution any time a CALL statement is issued.

FILE Default, same as omitting <type identifier>

Since some fields are dynamically loaded, data may not be available when a particular field table is executed, all fields are not available to all field tables. The following table summarizes which source of data fields may be used with each field table:

<u>SOURCE OF DATA</u>	<u>TYPE OF FIELD TABLE</u>			
	FILE	QDF	QUALIFY	CALL
Data base fields	X			
Data base fields extended	X			
Index fields	X			
Global definition	X	X	X	X
QDF		X	X	
QDF extended		X		
Qualify extension to QDT		X	X	

another source of data for the field table is in the field routine discussed below.

The CALL type of field table cannot have a FOR CLAUSE in the FIELD TABLE statement since the field table is executed on demand.

The type of FIELD TABLE QDF can be assigned a name. Only one FIELD TABLE QDF can be executed at any one time. However, the user is allowed to have more than one FIELD TABLE QDF defined in a GIPSY application. The command:

CHANGE FIELD TABLE QDF [TO <name>].

is available to change the current FIELD TABLE QDF to the requested one. If one has not been accessed, this command accesses the one specified. If a FIELD TABLE QDF has no name, this one is automatically accessed until it is changed. The "no-name" FIELD TABLE QDF can be reaccessed with the command "CHANGE FIELD TABLE QDF."

The FIELD TABLE QUALIFY can also be assigned a name. Only one FIELD TABLE QUALIFY can be executed at any one time. However, the user is allowed to have more than one FIELD TABLE QUALIFY defined in a GIPSY application. The command:

CHANGE FIELD TABLE QUALIFY [TO <name>].

is available throughout the GIPSY system for changing the current FIELD TABLE QUALIFY to the requested one. If one hasn't been accessed, this command accesses the one specified. If a FIELD TABLE QUALIFY has no name, the system automatically accesses this one until it is changed. The "no-name" FIELD TABLE QUALIFY can be reaccessed with the command "CHANGE FIELD TABLE QUALIFY."

C.3 The CALL in a Field Table

The CALL type of field table is executed by the user with the command:

```
CALL <name>.
```

This structure can be executed at any time a key word is expected during GIPSY provided the structure has been defined. This structure is especially useful for performing a task outside the realm of GIPSY by taking full advantage of the user subroutine capability. Consider the following example:

```
BUILD FDT.  
ADD.  
    SERVICE-CODE  1  A1.  
    EQUIP-CODE    2  A2.  
    REST-OF-DATA  3  A30.  
ADD TO GLOBAL.  
    USER-SERVICE *  A1.  
    USER-EQUIP   *  A1.  
END.  
//  
FIELD TABLE/CALL GET-USER-INFO.  
    $DISPLAY ("ENTER 1-CHARACTER CODE FOR REQUESTED SERVICE").  
    USER-SERVICE = $ACCEPT-USER-INPUT.  
    $DISPLAY ("ENTER 1-CHARACTER CODE FOR REQUESTED SERVICE").  
END.  
//  
CALL GET-USER-INFO.  
//  
RETRIEVE IF SERVICE-CODE EQ USER-SERVICE AND EQUIP-CODE EQ USER-EQUIP.
```

In the above example, the statement "CALL GET-USER-INFO" executes the FIELD TABLE CALL structure. During that execution, two messages are displayed on the terminal (\$DISPLAY) and two user inputs are accepted (fictional \$ACCEPT-USER-INPUT) and placed into GLOBAL fields. The RETRIEVE statement has access to those fields since all statements that reference fields have access to GLOBAL fields. In this example, only those records whose SERVICE-CODE and EQUIP-CODE fields match the ones requested by the user will be retrieved.

C.4 The Field Routine

The field routine is essentially a field table which can be used as a user call table subroutine. The body of the field routine as defined for a field table is composed of the syntax for the field routine as:

```
FIELD ROUTINE NAMED <name> [RETURN <return argument>] [<argument clause>]  
    [<execution control clauses>].  
    <field table executable statements>  
END ROUTINE.
```

There are six types of field routines available to the user: FILE, QDF, QUALIFY, CALL, GLOBAL, and OUTPUT. The FILE type is defaulted if no type is included. GLOBAL field routines may be referenced within any field table structure. The FILE, QDF, CALL and OUTPUT field routines may only be referenced within the field table for which they are valid:

<u>FIELD ROUTINE TYPE</u>	<u>AVAILABLE WITHIN</u>
FILE	FIELD TABLE
QDF	FIELD TABLE QDF, OUTPUT TABLE
QUALIFY	FIELD TABLE QUALIFY, FIELD TABLE QDF
CALL	FIELD TABLE CALL
OUTPUT	OUTPUT TABLE

The user can also define "arguments" in the opening sentence for all field routine types. These arguments are used to hold the data being passed to or from the field routine. Argument definition is done in an <argument clause>. There is no limit on the number of arguments which can be defined for a field routine. Each argument must consist of the following:

<argument type> <argument name>

where <argument type> defines the direction (from or to) of the data for use by the field routine and must be one of the following key words:

RETURN	Defines the area for accepting an <extended field> type of returned data
INPUT	Defines the area for holding data to be input to the field routine
OUTPUT	Defines the area for accepting data returned from the field routine
INOUT	Defines a data area that will be used as both an input and output location

The <argument name> is the name of a pre-defined field that is of a valid type for this field routine.

Below are some examples of field routine opening sentences and what they mean:

FIELD ROUTINE NAMED PROCESS-DATA-LIST RETURN DATA-AREA
(INPUT DATA-POINT INPUT DATA-STRING).

This field routine opening sentence has defined a field routine name PROCESS-DATA-LIST that may be referenced only within a FIELD TABLE. The key word "RETURN" denotes that a field called "DATA-AREA" will be used to hold

data returned from execution of this field routine. Two input arguments have been defined: DATA-POINT and DATA-STRING. The fields DATA-AREA, DATA-POINT, and DATA-STRING can be either FILE, EXTENDED, or GLOBAL fields.

FIELD ROUTINE QDF NAMED BUILD-NEW-DATE RETURN NEW-DATE
(INPUT OLD-DATE, OUTPUT SEASON).

This field routine statement has defined a field routine name BUILD-NEW-DATE that may be referenced only within FIELD TABLE QDF or OUTPUT TABLE. The field "NEW-DATE" has been defined to hold the returned data after execution. An input argument, OLD-DATE, has been defined that will accept data passed to the field routine. A field named SEASON has been defined as the location to hold a second output from execution of this field routine. The fields NEW-DATE, OLD-DATE, and SEASON can be either QDF, QDF-extended, or GLOBAL fields.

FIELD ROUTINE QUALIFY NAMED BUILD-QUALIFIED-EQUIPMENT-FIELD.

This statement has defined a field routine that has no arguments and can be referenced only within the FIELD TABLE QUALIFY or FIELD TABLE QDF.

FIELD ROUTINE CALL NAMED GET-USER-TIME-INPUT RETURN USER-TIME
(INPUT AM-PM).

This statement has defined a field routine that may be referenced only within a FIELD TABLE CALL. The field USER-TIME has been denoted to hold the returned data after execution. The field AM-PM serves as both an input and output area. The fields USER-TIME and AM-PM can only be GLOBAL fields.

FIELD ROUTINE OUTPUT NAMED BUILD-OUTPUT-ELEMENT
(INPUT OUTPUT-ONE INPUT OUTPUT-TWO).

This field routine may only be referenced within a substructure of the Output Table. The two input fields, OUTPUT-ONE and OUTPUT-TWO may be QDF, QDF extended, QUALIFY, or GLOBAL fields.

FIELD ROUTINE GLOBAL NAMED GET-CURRENT-TIME
RETURN = CURRENT-TIME.

This field routine opening sentence has defined a field routine named GET-CURRENT-TIME and may be referenced within any field table. The field CURRENT-TIME will contain the returned data after execution. This field can only be a GLOBAL field.

The fields referenced within a field routine must be defined in the "BUILD FDT" structure before they can be used. The field types (QDF, FILE, etc.) must be compatible with the field routine type.

Before one field routine can be referenced in another, it must have been previously defined. The internal field routine type must match the type of the routine referencing it. The only exception to this rule is the FIELD

ROUTINE GLOBAL which may be referenced by any field table.

All other types of data manipulation outlined in section C.1.2.2 are available within the field routine with no other exceptions or constraints.

The primary function of the field routine capability is to allow users to break their field tables into easily understood and developed functional blocks. It serves a secondary function in that GIPSY applications become more documentable and easier to maintain.

Below is an example of the definition and use of several field routines:

BUILD FDT.

ADD.

SERVICE-CODE	1	A1.	
EQUIP-CODE	2	A2.	
NBR-EQUIP	3	I5.	
READY-PCNT	8	I3.	
SVC-TOTAL-EQ	*	I24.	
ARMY-TOTAL	SVC-TOTAL-EQ(1)		* I6.
AIR-TOTAL	SVC-TOTAL-EQ(7)		* I6.
MARINE-TOTAL	SVC-TOTAL-EQ(13)		* I6.
NAVY-TOTAL	SVC-TOTAL-EQ(19)		* I6.
SERVICE-PCNT	*	I16.	
ARMY-PCNT	SERVICE-PCNT(1)		* I4.
AIR-PCNT	SERVICE-PCNT(5)		* I4.
MARINE-PCNT	SERVICE-PCNT(9)		* I4.
NAVY-PCNT	SERVICE-PCNT(13)		* I4.
SERVICE-AVG	*	I12.	
ARMY-AVG	SERVICE-AVG(1)		* I3.
AIR-AVG	SERVICE-PCNT(4)		* I3.
MARINE-AVG	SERVICE-AVG(7)		* I3.
NAVY-AVG	SERVICE-AVG(10)		* I3.
SERVICE-RECS	*	I12.	
ARMY-NR-RECS	SERVICE-RECS(1)		* I3.
AIR-RECS	SERVICE-RECS(4)		* I3.
MARINE-RECS	SERVICE-RECS(7)		* I3.
NAVY-RECS	SERVICE-RECS(10)		* I3.
SVC-TOT-EQ	*	I6.	
SVC-EQ-PCNT	*	I4.	
TEMP-PERCENT	*	I4.	
SVC-EQ-RECS	*	I3.	
NBR-SVC-RECS	*	I4.	

END.

//

FIELD ROUTINE NAMED COMPUTE-AVERAGE-PERCENT (RETURN = SVC-EQ-AVG,
INPUT = NBR-SVC-RECS, INOUT = SVC-EQ-PCNT).
ADD READY-PCNT TO SVC-EQ-PCNT.
TEMP-PERCENT = SVC-EQ-PCNT.
DIVIDE TEMP-PERCENT BY NBR-SVC-RECS.

```

    SVC-EQ-AVG = TEMP-PERCENT.
END ROUTINE.
//
FIELD ROUTINE NAMED TOTAL-EQUIPMENT-AND PERCENT (INOUT = SVC-TOT-EQ,
    INOUT = SVC-EQ-PCNT, INOUT = SVC-EQ-AVG, INOUT = NBR-SVC-RECS).
    ADD NBR-EQUIP TO SVC-TOT-EQ.
    ADD 1 TO NBR-SVC-RECS.
    SVC-EQ-AVG = $$COMPUTE-AVERAGE-PERCENT (NBR-SVC-RECS, SVC-EQ-PCNT).
END ROUTINE.
//
FIELD TABLE FOR INITIAL.
    SVC-TOTAL-EQ = 0.
    SERVICE-PCNT = 0.
    SERVICE-AVG = 0.
    SERVICE-RECS = 0.
END.
//
FIELD TABLE.
    $$TOTAL-EQUIPMENT-AND-PERCENT (ARMY-TOTAL, ARMY-PCNT, ARMY-AVG,
        ARMY-NR-RECS) IF SERVICE-CODE EQ A.
    $$TOTAL-EQUIPMENT-AND-PERCENT (AIR-TOTAL, AIR-PCNT, AIR-AVG,
        AIR-NP-RECS) IF SERVICE-CODE EQ F.
    $$TOTAL-EQUIPMENT-AND-PERCENT (MARINE-TOTAL, MARINE-PCNT, MARINE-AVG,
        MARINE-NR-RECS) IF SERVICE-CODE EQ M.
    $$TOTAL-EQUIPMENT-AND-PERCENT (NAVY-TOTAL, NAVY-PCNT, NAVY-AVG,
        NAVY-NR-RECS) IF SERVICE-CODE EQ N.
END.

```

In this example, the field table executes a field routine. Notice the order in which the routines were defined. A field routine must have already been defined before it can be referenced within another field table.

The following PCS (figure C-1) demonstrates the usage of several GIPSY subroutines used in the GDRMOD module. The command "DISPLAY BOOK PLACE *." produced the first output (figure C-2). A subsequent command "LOCATION" displayed the first page of the "part" named LOCATION (figure C-3).


```

1400 END.
1410 END.
1420 //
1430 HEADER TABLE PART 2.
1440 FIELD TABLE WHEN PLACE-TYPE EQ LOCATION.
1450 -PAGE-1 SPACE-NUMBER 2> PIC -22,222- OF -TOTAL-PAGES 2>
1460 LINE PIC -22,222- COL 65 LEFT JUSTIFIED TEXT COMPRESSED.
1470 SPACE 1.
1480 LINE -GEOGRAPHIC LOCATIONS AVAILABLE IN GIPSY-CENTERED.
1490 SPACE 2.
1500 END.
1510 //
1520 BODY TABLE PART 2.
1530 FIELD TABLE WHEN PLACE-TYPE EQ LOCATION.
1540 LINE PLACE-NAME(1/30) COORD-1 CENTERED.
1550 SPACE 1 IF PLACE-NAME(1/1) COMPLETE AND $LINES-LEFT(2) GE 3 OR
1560 $LINES-LEFT(2) EQ 0.
1570 EJECT IF PLACE-NAME(1/1) COMPLETE AND $LINES-LEFT(2) LT 3.
1580 END.
1590 //
1600 TRAILER TABLE PART 2.
1610 FIELD TABLE WHEN PLACE-TYPE EQ LOCATION.
1620 SPACE 1.
1630 LINE -2 2, L O C A T I O N S 2 2 2- CENTERED.
1640 END.
1650 END.
1660 END OUTPUT.
1670 BUILD OUTPUT REPORT TO PLACE.

```

Figure C-1. PCS With GIPSY Subroutines (Part 2 of 2)

GEOGRAPHIC WINDOWS AVAILABLE IN GIPSY

AFARS AND ISSAS	105403N0414549E	124745N0432216E
AFGHANISTAN	292353N0672925E	382935N0745138E
AFRICA	345246S0172920W	372352N0573934E
ALABAMA	301236N0882930W	350131N0645427W
ALASKA	525100N1685556W	712048N1295926W
ALBANIA	393756N0191641E	423942N0210333E
ALDABRA ISLAND	092748S0461131E	091956S0463118E
ALGERIA	185913N0083355W	370747N0114245E
ANDORIA	422629N0012706E	424003N0014830E
ANDORRA	422629N0012706E	424003N0014830E
ANGOLA	180200S0113416E	042450S0241115E
ANTARCTICA	870000S1795959W	600000S1899999E
ANTARCTICA	870000S1795959W	600000S1899999E
ANTIGUA	165922N0615330W	171003N0613958W
ARGENTINA	552207S073232W	214710S0534054W
ARIZONA	311918N1144945W	370027N1090329W
ARKANSAS	330006N0943734W	363005N0993944W
AUSTRALIA	433630S1130831E	104221S1533340E
AUSTRIA	462335N0092753E	490106N0171011E
BAHAMA ISLANDS	205305N0792011W	271413N0710607W
BAHAMAS	205305N0792011W	271413N0710607W
BAHRAIN	254712N0502654E	261430N0503741E
BANGLADESH	204336N0880226E	263559N0924095E
BELGIUM	492948N0023205E	513010N0052404E
BERMUDA	321537N0645237W	322258N0643217W
BHUTAN	264050N0984428E	281634N0982259E

W I N D O W S

Figure C-2. Output From BOOK-PLACE, PART-WINDOW

GEOGRAPHIC LOCATIONS AVAILABLE IN GIPSY

ALBANY	434000N0734500W
ALBUQUERQUE	350500N1054700W
AMARILLO	351100N1015000W
ANCHORAGE	611300N1495400W
ATLANTA	334500N0842300W
ATLANTIC CITY	392200N0742500W
AUSTIN	392900N1170400W
BAKER	444700N1175000W
BALTIMORE	391800N0763800W
BISMARCK	464800N1004700W
BOISE	433600N1161300W
BOSTON	422100N0710500W
BUFFALO	425500N0785000W
CALGARY	510100N1140100W
CARLSBAD	322600N1041500W
CHARLESTON SC	324700N0795600W
CHARLESTON WVA	382100N0813800W
CHARLOTTE	351400N0805000W
CHEYENNE	410900N1045200W
CHICAGO	415000N0873700W
CINCINNATI	390800N0843000W
CLEVELAND	412800N0813700W
COLUMBIA	340000N0810200W
COLUMBUS	400000N0830100W

*** LOCATIONS ***

Figure C-3. Output From BOOK-PLACE, PART-LOCATION

THIS PAGE INTENTIONALLY LEFT BLANK .

APPENDIX D
BUILD NEW REPORT EXAMPLE

THIS PAGE INTENTIONALLY LEFT BLANK

The following terminal session demonstrates the usage of the DEFINE/ADD and INPUT commands discussed in section 3.3.4.6, Manual Data Input. Recall the syntax of each command:

```
{ DEFINE } { <vector mode> named <vector name list> }
{ ADD } { <number of headers> <vector mode> } .

INPUT [ <vector mode> ] [ <vector name list> ] [ <order> ] ; VALUES ARE <value list>
ARE ALL <value>
ARE FROM <value> [ BY <value> ] .

Where <order> is defined as FOR vector mode [ <vector list> ] ; [ :.. ]
```

In this session we will build a new tabular report with 3 columns and 2 rows, later add an additional column and row, and finally expand the report to include categories and sections. The following conventions will be used to identify an element with its proper placement in the report matrix. Each element is of the form

```
cr      or      cr.ca

where c = column number
      r = row number
      c = category number
      s = section number.
```

The final report will appear as follows:

BUILD NEW REPORT EXAMPLE

	CATEGORY 1				CATEGORY 2			
	FIRST	SECOND	THIRD	FOURTH	FIRST	SECOND	THIRD	FOURTH
SECTION ONE								
ROW 1	11.11	21.11	31.11	41.11	11.21	21.21	31.21	41.21
ROW 2	12.11	22.11	32.11	42.11	12.21	22.21	32.21	42.21
ROW 3	13.11	23.11	33.11	43.11	13.21	23.21	33.21	43.21
SECTION TWO								
ROW 1	11.12	21.12	31.12	41.12	11.22	21.22	31.22	41.22
ROW 2	12.12	22.12	32.12	42.12	12.22	22.22	32.22	42.22
ROW 3	13.12	23.12	33.12	43.12	13.22	23.22	33.22	43.22

Throughout the session, a sequence of commands with comments will be presented along with the resulting report. The comments are intended to not only outline the report development but also to point out potential errors.

We begin by entering the GIPSY Display mode with the GIPSYD command. A GDS is normally required. However, we are going to manually create a new report, typing in the data from the keyboard. In this case a carriage return may be given in response to "Type in cat/fil string of your Graphic Data Set (GDS)".

```
*GIPSYD
GIPSY Release 3.1.1 (7 Mar 81) ***PRODUCTION SYSTEM***
For any problems or questions about this release
call the GIPSY support office at x53395 (A/V 225-3395)
Type in cat/fil string of your Graphic Data Set (GDS)
>
***WARNING*** No Graphic Data Set (GDS) is attached. You must access
your GDS before attempting to display any tabular or graphic report.
>BUILD NEW REPORT.
>REVIEW.
```

D-5

NO GRAPHIC DATA SET SUPPLIED 1 OF 1

NODATA 0
NODATA 0

Note that the BUILD NEW REPORT command automatically creates a report with one row and one column entitled "NO GRAPHIC DATA SET SUPPLIED". We wish to change the title to "BUILD NEW REPORT EXAMPLE". Since the TITLE statement is not valid in Build New Report mode, we must exit to input the title. We then reenter Build New Report mode to define our columns and rows. To exit the Build New Report mode enter one of the following commands:

```
>END.
OR
>END BUILD NEW REPORT.
```

```

>TITLE "BUILD NEW REPORT EXAMPLE".
>BUILD NEW REPORT.
>DEFINE COLS NAMED FIRST,SECOND,THIRD.
>DEFINE 2 ROWS.
>REVIEW.

```

We have defined 3 columns and 2 rows, allowing GIPSY to supply the default row headers "ROW 1" and "ROW 2". The resulting report is shown below. Notice that the DEFINE command deleted the column named "NO DATA" and row named "NO DATA" which were present and replaced them with columns FIRST, SECOND, THIRD and rows ROW 1 and ROW 2. All of the report values are initialized to zeroes.

1 OF 1

BUILD NEW REPORT EXAMPLE

	FIRST	SECOND	THIRD
ROW 1	0	0	0
ROW 2	0	0	0

We now will present several ways to use the INPUT command to replace the zero values with the desired data. One method is to specify a particular row or column by its header name.

```
>INPUT ROW 'ROW 1';VALUES ARE 11,21,31.
>INPUT ROW 'ROW 2';VALUES ARE 12,22,32.
>ENDVIEW.
```

1 OF 1

BUILD NEW REPORT EXAMPLE

	FIRST	SECOND	THIRD
ROW 1	11	21	31
ROW 2	12	22	32

Note the placement of the semicolons after the header names above. The semicolon is required to indicate the end of a header list. Below we combined the two input statements and illustrate the message displayed when the semicolon is inadvertently left out.

```
>INPUT ROW "ROW 1","ROW 2" VALUES ARE 11,21,31,12,22,32.
The item below is not recognized.
Expecting the next header name in the list or
the rest of the range of Headers (thru <header-name>). (1724)
INPUT ROW "ROW 1","ROW 2" VALUES ARE 11,21,31,12,22,32.
>VALUES
```

Header names need not be specified with the INPUT command. Therefore, another equivalent statement might be:

>INPUT ROWS VALUES ARE 11,21,31,12,22,32.

Not only are header names not required, neither is the vector mode ROW. However, care must be taken to ensure the values are entered in the desired order.

>INPUT VALUES ARE 11,21,31,12,22,32.

D-8

1 OF 1

BUILD NEW REPORT EXAMPLE

	FIRST	SECOND	THIRD
ROW 1	11	31	22
ROW 2	21	12	32

Since no vector mode was specified above, the command caused the values to be placed in the report in column order which is the default. We previously entered the values in row order. Recall, the default ordering is column, row, category section. To change the values back the correct command would be:

>INPUT VALUES ARE 11,12,21,22,31,32.

>REVIEW.

There is still another alternative for entering the same values. If no specific values are supplied in the INPUT statement, GIPSY will prompt you for each value, notifying you when too few or too many values have been typed. When using the prompting sequence DO NOT end each line of values with a period, this will end the INPUT process.

```
>INPUT VALUES.
INPUT 1 VALUES FOR COL  FIRST
>11,12
INPUT 2 VALUES FOR COL  SECOND
>21
INPUT 1 MORE VALUES FOR COL  SECOND
>22
INPUT 2 VALUES FOR COL  THIRD
>31,32,33
value list exceeds vector length enter period or
enter a carriage return to delete the entire statement; then retype (175?)
```

```
INPUT VALUES.
11,12
21
22
31,32,33
^
>.
value list exceeds vector length enter period or
enter a carriage return to delete the entire statement; then retype (175?)
```

```
INPUT VALUES.
11,12
21
22
31,32,33
^ ^
>.
REVIEW.
```

This concludes the portion of the session illustrating the creation of the basic report, which we will expand.

We will now use the ADD command to append a new row and a new column. GIPSY supplies the name "ROW 3" for the additional row.

```
>ADD 1 FCM.
>FEVIEW.
```

1 OF 1

BUILD NEW REPORT EXAMPLE

	FIRST	SECOND	THIRD
ROW 1	11	21	31
ROW 2	12	22	32
ROW 3	0	0	0

Notice that like the DEFINE command, the ADD command initializes new row and new column values to zero.

Suppose we now wish to use the ALL ARE value option to enter the value 13 across the entire new row, "ROW 3". The command would be:

```
>INPUT ROW 'ROW 3';VALUES ARE ALL 13.
>FEVIEW.
```

The resulting report is on the next page.

BUILD NEW REPORT EXAMPLE

1 of 1

	FIRST	SECOND	THIRD
ROW 1	11	21	31
ROW 2	12	22	32
ROW 3	13	13	13

Now we wish to replace the last 2 values in "ROW 3" with the values 23 and 33. We will employ the ordering option to specify these elements. Note the placement of the semicolons.

>INPUT ROW 'ROW 3';FOR COLS SECOND,THIRD;VALUES FIRE 23,33.
>REVIEW.

BUILD NEW REPORT EXAMPLE

1 of 1

	FIRST	SECOND	THIRD
ROW 1	11	21	31
ROW 2	12	22	32
ROW 3	13	23	33

The values 23, 33 replaced the old values 13, 13 in ROW 3 columns SECOND and THIRD as desired.

We will now add a new column called FOURTH (what an original name).

>ADD COL NAMED FOURTH.
>REVIEW.

1 OF 1

BUILD NEW REPORT EXAMPLE

	FIRST	SECOND	THIRD	FOURTH
ROW 1	11	21	31	0
ROW 2	12	22	32	0
ROW 3	13	23	33	0

Again we see the values in the new column are initialized to zero. Next, we will enter the values into column FOURTH in reverse order (from "ROW 3" to "ROW 1") using the vector mode THRU vector mode option. We will also illustrate the FROM value BY value option to generate the values 43, 42, 41.

D-12

>INPUT COL FOURTH;FOR ROWS 'ROW 3' THRU 'ROW 1';VALUES ARE FROM 43 BY -1.
>REVIEW.

1 OF 1

BUILD NEW REPORT EXAMPLE

	FIRST	SECOND	THIRD	FOURTH
ROW 1	11	21	31	41
ROW 2	12	22	32	42
ROW 3	13	23	33	43

We are now ready to append an additional section and an additional category.

```
>ADD SECTION NAME 'SECTION TWO'.
>FVIEW.
>ADD CATEGORY NAME 'CATEGORY 2'.
>FVIEW.
```

1 OF 1

BUILD NEW REPORT EXAMPLE

CATEGORY 2

	FIRST	SECOND	THIRD	FOURTH	FIRST	SECOND	THIRD	FOURTH
ROW 1	11	21	31	41	0	0	0	0
ROW 2	12	22	32	42	0	0	0	0
ROW 3	13	23	33	43	0	0	0	0

SECTION TWO

ROW 1	0	0	0	0	0	0	0	0
ROW 2	0	0	0	0	0	0	0	0
ROW 3	0	0	0	0	0	0	0	0

NOTE: The first section and first category are not labeled with vector header names. They may be referenced as SECTION " " and CATEGORY " " or may be renamed with the RENAME command. Here we will rename them for clarity.

```
>DEFINE SECTION " " NAME "SECTION ONE".
>DEFINE CATEGORY " " NAME "CATEGORY 1".
```

Errors can be made easily when entering values into a report containing categories and sections. The best ways to ensure proper placement of values in the report are

1. When entering values for an entire section, category or report, allow GIPSY to prompt you for input by typing one of the following

```
INPUT SECTION[<section name list>]VALUES.
or INPUT CATEGORY[<category name list>] VALUES.
or INPUT VALUES.
```

D-14

2. When adding to or modifying single values in a particular SECTION & CATEGORY of an existing report, specify all four vector modes. For example, to revise the first entry in our report to 11.11 we might type

```
INPUT ROW "ROW 1"; FOR COL FIRST; FOR SECTION "SECTION ONE"; FOR CATEGORY "CATEGORY 1"; VALUES ARE 11.11.
```

Of course, those users very familiar with the Build New Report structure need not limit themselves to the above suggestions.

To reflect the addition of categories and sections we will change the entries in "CATEGORY 1" by specifying only the category name and allowing GIPSY to prompt us for input.

```
>INPUT CATEGORY "CATEGORY 1" ;VALUES.
IN SECTION SECTION ONE ,ROW ROW 1 ,INPUT 4 VALUES FOR CATEGORY CATEGORY 1
>11.11,21.11,31.11,41.11
IN SECTION SECTION ONE ,ROW ROW 2 ,INPUT 4 VALUES FOR CATEGORY CATEGORY 1
>12.11,22.11,32.11,42.11
IN SECTION SECTION ONE ,ROW ROW 3 ,INPUT 4 VALUES FOR CATEGORY CATEGORY 1
>12.11,23.11,33.11,43.11
IN SECTION SECTION TWO ,ROW ROW 1 ,INPUT 4 VALUES FOR CATEGORY CATEGORY 1
>11.12,21.12,31.12,41.12
IN SECTION SECTION TWO ,ROW ROW 1 ,INPUT 4 VALUES FOR CATEGORY CATEGORY 1
>12.12,22.12,32.12,42.12,13.12,23.12,33.12,43.12.
>FEVIEW.
```

BUILD NEW REPORT EXAMPLE					1 OF 1					
					CATEGORY 2					
					CATEGORY 1					
					FIRST	SECOND	THIRD	FOURTH	FIRST	SECOND
SECTION ONE										
ROW 1					11.11	21.11	31.11	41.11	0.00	0.00
ROW 2					12.11	22.11	32.11	42.11	0.00	0.00
ROW 3					13.11	23.11	33.11	43.11	0.00	0.00
SECTION TWO										
ROW 1					11.12	21.12	31.12	41.12	0.00	0.00
ROW 2					12.12	22.12	32.12	42.12	0.00	0.00
ROW 3					13.12	23.12	33.12	43.12	0.00	0.00

Notice that instead of going through the entire prompting sequence we entered all of "SECTION TWO" values at once.

Finally, we will enter new values to replace the remaining zeroes. We allow GIPSY to prompt us for "SECTION ONE", "CATEGORY 2" values and then the "SECTION TWO", "CATEGORY 2" values. Each time we end the prompting sequence early because we know what values will be asked for in succeeding prompts.

```
>INPUT SECTION "SECTION ONE";FOR CATEGORY "CATEGORY 2";VALUES.
IN ROW ROW 1 ,COL FIRST ,INPUT 1 VALUES FOR SECTION SECTION ONE
>11.21
IN ROW ROW 1 ,COL SECOND ,INPUT 1 VALUES FOR SECTION SECTION ONE
>21.21
IN ROW ROW 1 ,COL THIRD ,INPUT 1 VALUES FOR SECTION SECTION ONE
>31.21
IN ROW ROW 1 ,COL FOURTH ,INPUT 1 VALUES FOR SECTION SECTION ONE
>41.21
IN ROW ROW 2 ,COL FIRST ,INPUT 1 VALUES FOR SECTION SECTION ONE
>12.21,22.21,31.21,42.21
IN ROW ROW 3 ,COL SECOND ,INPUT 1 VALUES FOR SECTION SECTION ONE
>13.21,23.21,33.21,43.21.
```

Now for Section Two:

```
>INPUT ROWS FOR "SECTION TWO";FOR CATEGORY "CATEGORY 2";VALUES.
IN ROW ROW 1 ,COL FIRST ,INPUT 1 VALUES FOR CATEGORY CATEGORY 2
11.22,21.22,31.22,41.22,12.22,22.22,32.22,42.22,13.22,23.22,33.22,43.22.
>REVIEW.
```

The final report is on the next page.

BUILD NEW REPORT EXAMPLE

		CATEGORY 1				CATEGORY 2			
		FIRST	SECOND	THIRD	FOURTH	FIRST	SECOND	THIRD	FOURTH
SECTION ONE									
ROW 1		11.11	21.11	31.11	41.11	11.21	21.21	31.21	41.21
ROW 2		12.11	22.11	32.11	42.11	12.21	22.21	32.21	42.21
ROW 3		13.11	23.11	33.11	43.11	13.21	23.21	33.21	43.21
SECTION TWO									
ROW 1		11.12	21.12	31.12	41.12	11.22	21.22	31.22	41.22
ROW 2		12.12	22.12	32.12	42.12	12.22	22.22	32.22	42.22
ROW 3		13.12	23.12	33.12	43.12	13.22	23.22	33.22	43.22

>END NEW REPORT.

>DONE.

This concludes our Build New Report session.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E

AREA NAMES

THIS PAGE INTENTIONALLY LEFT BLANK

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
AFARS AND ISSAS	105403N0414949E	124745N0432216E	
AFGHANISTAN	292355N0602925E	382935N0745138E	C-AF
AFRICA	345246S0172920W	372352N0573934E	
ALABAMA	301236N0882930W	350131N0845427W	C-01
ALASKA	525100N1685556W	712048N1295926W	C-02
ALBANIA	393756N0191641E	423942N0210330E	C-AL
ALDABRA ISLAND	092748S0461131E	091956S0463118E	
ALGERIA	185913N0083355W	370747N0114245E	C-AG
ANDORRA	422629N0012706E	424003N0014830E	C-AN
ANGOLA	180200S0113416E	042450S0241115E	C-AO
ANTARCTICA	870000S1795959W	600000S180000E	C-AY
ANTIGUA	165922N0615330W	171003N0613958W	C-AC
ARGENTINA	552207S0732323W	214710S0534054W	C-AR
ARIZONA	311918N1144945W	370027N1090329W	C-04
ARKANSAS	330006N0943734W	363005N0893944W	C-05
AUSTRALIA	433630S1130831E	104221S1533340E	C-AS
AUSTRIA	462335N0092753E	490106N0171011E	C-AU
BAHAMAS	205305N0792011W	271413N0710607W	C-BF
BAHRAIN	254712N0502654E	261430N0503741E	C-BA
BANGLADESH	204336N0880226E	283559N0024006E	C-BG
BARBUDA	173021N0620007W	174440N0613515W	
BELGIUM	402948N0023205E	513010N0062404E	C-BE

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
BELIZE	155228N0891315W	182930N0880437W	C-BD
BERMUDA	321537N0645237W	322258N0643817W	C-BT
BHUTAN	264050N0884428E	281634N0920858E	
BOUVIA	225503S0693939W	094047S0572708W	C-BL
BOTSWANA	265356S0195835E	174456S0292233E	C-BC
BR INDIAN OCEAN TERRITORY	092748S0461131E	091956S0463118E	
BRAZIL	335039S0740531W	051706N0343711W	C-BR
BRIT HONDURAS	155228N0891315W	182930N0880437W	
BRIT INDIAN OCEAN TERRITORY	092748S0461131E	091956S0463118E	C-IO
BRITISH HONDURAS	155228N0891315W	182930N0880437W	C-BH
BRUNEI	040217N1140022E	050437N1152510E	C-BX
BULGARIA	411345N0222059E	441403N0283701E	C-BU
BURMA	100147N0920931E	283016N1010857E	C-BM
BURUNDI	042756S0285847E	021740S0305120E	C-BY
CALIFORNIA	323135N1242521W	420051N1140845W	C-06
CAMBODIA KHMER REPUBLIC	102606N1021854E	144224N1073611E	C-CB
CAMEROON	013704N0082855E	130629N0161134E	C-CM
CANADA	410653N1411048W	742837N0523827W	C-CA
CANAL ZONE	085243N0800506W	092327N0792041W	C-PO
CANTON AND ENDERBURY ISLANDS	024711S1730508W	024511S1730348W	C-EQ
CAPE VERDE ISLANDS	144713N0252222W	171158N0223933W	C-CU
CARIBBEAN SEA	075304N08902228W	215022N0605719W	C-IX

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
CENTRAL AFRICAN REPUBLIC	021131N0142240E	110105N0273049E	C-CF
CHAD	072505N0132717E	233137N0240139E	C-CD
CHILE	555102S0755237W	173036S0663453W	C-CI
CHINA	181130N0733918E	532657N1345059E	C-CH
COLOMBIA	041535S0790611W	122731N0665133W	C-CC
COLORADO	365947N1090322W	410019N1020103W	C-CB
COMORES ISLANDS	130014S0431218E	112146S0451754E	C-CF
CONGO-BRAZZAVILLE	050341S0110705E	034340N0183903E	C-09
CONNECTICUT	410027N0734343W	420322N0714729W	
CONUS	P45721N1244326W	492239N0665759W	
CORSE	412132N0083255E	430045N06093210E	
COSTA RICA	080059N0855622W	111314N0823316W	C-CS
CUBA	194732N0845904W	231221N0740705W	C-CU
CYPRUS	343327N0321551E	354119N0343513E	C-CY
CZECHOSLOVAKIA	474636N0115446E	510827N0222313E	C-CZ
D C	384724N0770717W	385942N0765408W	
DAHOMEY	061218N0004506E	122349N0035127E	
DELAWARE	382700N0754713W	395119N0750218W	C-10
DENMARK	543320N0080401E	574413N0151042E	C-DA
D. C.	384724N0770717W	385942N0765408W	C-11
DJIBOUTI	105537N0414522E	124221N0432526E	C-DJ

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
DOMINICA	151144N0612931W	153757N0611500W	C-DC
DOMINICAN REPUBLIC	173605N0720033W	195553N0681910W	C-DR
EAST GERMANY	500946N0095154E	544117N0150201E	C-EC
ECUADOR	050132S0810102W	012457N0751251W	C-EG
EGYPT	214223N0244217E	313658N0354851E	C-ES
EL SALVADOR	130833N0900704W	142717N0874144W	C-EK
ELLICE ISLANDS	084130S1770640E	070943S1791512E	C-ET
EQUATORIAL GUINEA	005941N0092123E	022025N0112113E	C-FA
ETHIOPIA	032500N0325812E	180045N0475956E	C-FJ
EUROPE	342300N0103453W	710835N0640930E	C-FI
FALKLAND ISLANDS	522627S0611920W	511427S0574347W	C-12
FAROE ISLANDS	612301N0073910W	622351N0061319W	
FII IS	195459S1764705E	155945S1781210W	
FINLAND	594401N0193514E	700508N0313419E	
FLORIDA	245721N0873757W	310006N0800121W	
FR SOUTHERN AND ANTARCTIC IS	494134S0685115E	483747S0703833E	
FRANCE	421942N0044931W	510435N0081437E	
FRENCH GUIANA	020624N0543615W	054711N0513923W	
FRENCH POLYNESIA	231136S1513814W	084303S1353821W	
FRENCH TERRITORY OF THE AFARS	105403N0414949E	124745N0432216E	
GABON	035522S0084136E	021932N0143144E	
GAMBIA	130332N0185001W	134930N0134747W	

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
GEORGIA	302116N0853740W	350018N0805046W	C-13
GERMANY FEDERAL REPUBLIC OF	471618N0055254E	550219N0134814E	C-GE
GERMANY SOVIET ZONE OF	500946N0095154E	544117N0150201E	C-GC
GHANA	044314N0031537W	110931N0011206E	C-GH
GILBERT ISLANDS	024029S1724219E	031605N1765120E	C-GS
GREECE	345425N0193751E	414430N0281253E	C-GR
GREENLAND	594956N0725258W	833646N0121206W	C-GL
GRENADA	115704N0620124W	121631N0612504W	
GUADELOUPE	155029N0614751W	163054N0605928W	C-GP
GUAM	131403N1443721E	133915N1445715E	C-GQ
GUATEMALA	134407N0921520W	174923N0881332W	C-GT
GUINEA	071059N0150610W	123934N0073914W	C-GU
GUINEA BISSAU	105516N0164316W	124105N0133829W	
GULF OF MEXICO	180000N0980000W	310000N0810000W	C-1M
GUYANA	011107N0612439W	082415N0562827W	C-GY
HAITI	180102N0742914W	200526N0713800W	C-HA
HAWAII	185529N1601525W	221425N1544731W	C-15
HENDERSON ISLAND	242448S1282051W	241930S1281708W	
HONDURAS	125847N0892140W	160131N0830809W	C-HO
HONG KONG	220832N1134919E	223640N1143007E	C-HK
HUNGARY	454812N0160250E	483419N0224622E	C-HU
HUNGARY	104202N0874143W	150049N0830758W	C-NU

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
ICELAND	632347N0243003W	663209N0133536W	C-1C
IDAHO	415916N1171325W	490037N1110339W	C-16
ILLINOIS	365718N0913200W	423037N0873145W	C-17
INDIA	080245N0680336E	365049N0972017E	C-IN
INDIAN OCEAN	380000S0250000E	250000N1150000E	C-6A
INDIANA	374619N0880637W	414612N0844658W	C-18
INDONESIA	105905S0945858E	055312N1412853E	C-ID
IOWA	403419N0963905W	433005N0900825W	C-19
IRAN	250349N0440259E	394554N0631819E	C-IR
IRAQ	290515N0384552E	372317N0483245E	C-IZ
IRELAND	512000N0104000W	553000N0052000W	C-EI
ISRAEL	293007N0341623E	331632N0353958E	C-IS
ITALY	375417N0063704E	470538N0182953E	C-IT
IVORY COAST	042000N0083626W	104321N0023019W	C-IU
JAMAICA	174138N0782247W	183115N0761124W	C-JM
JAN MAYEN	704814N0090725W	711052N0075545W	
JAPAN	241202N1233932E	452957N1454757E	C-JA
JORDON	291050N0345253E	332209N0391731E	C-JO
KANSAS	365942N1020157W	400019N0943653W	C-20
KENTUCKY	363001N0892614W	390810N0815752W	C-21
KENYA	044037S0335424E	050131N0415510E	C-KE

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
KERGUELEN	494331S0683616E	483357S0703405E	
KOREA NORTH	374052N1241952E	425451N1305019E	C-KN
KOREA, REPUBLIC OF	330853N1260821E	383024N1305829E	C-KS
KUWAIT	283353N0463440E	300517N0481940E	C-KU
LA REUNION ISLAND	212439S0550238E	205340S0554220E	
LAOS	135014N1001223E	223152N1072814E	C-LA
LEBANON	330333N0350553E	344126N0363731E	C-LE
LESOTHO	303929S0270042E	283353S0292728E	C-LT
LIBERIA	041959N0112933W	083412N0072229W	C-LI
LIBYA	193036N0090923E	330931N0250150E	C-LY
LIECHTENSTEIN	470324N0092804E	471629N0093810E	C-LS
LOUISIANA	285456N0940326W	330033N0890020W	C-22
LUXEMBOURG	492642N0054351E	501052N0063129E	C-LU
MADAGASCAR	253646S0430812E	121029S0501532E	C-MA
MAINE	430341N0710533W	472822N0665759W	C-23
MALAU	170942S0324041E	092255S0355451E	C-MI
MALAYSIA	005534N1001222E	065704N1191620E	C-MY
MALI	101212N0120720W	250111N0040937E	C-ML
MALTA	354758N0141047E	360430N0143451E	C-MT
MARTINIQUE	142404N0611356W	145252N0604835W	C-MB
MARYLAND	375353N0792944W	394313N0750213W	C-24
MASSACHUSETTS	411438N0733103W	425327N0695529W	C-25

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
MAURITANIA	144809N0165541W	271625N0045133W	C-MR
MAURITIUS	203304S057160SE	195750S0574932E	C-MP
MEDITERRANEAN SEA	301258N0055140W	454011N0360916E	
MEXICO	143527N1181420W	324030N0864749W	C-MX
MICHIGAN	414120N0902503W	472930N0822516W	C-26
MIDDLE EAST	440000N0240000E	200000N0760000E	
MIDWAY ISLANDS	281018N1772549W	281620N1771827W	C-MQ
MINNESOTA	432850N0971443W	492239N0893450W	C-27
MISSISSIPPI	301010N0913941W	350017N0880516W	C-28
MISSOURI	355852N0954640W	403736N0890658W	C-29
MONACO	433832N0071044E	435145N0073658E	
MONGOLIA	413611N0874910E	521005N1192656E	C-MG
MONTANA	442120N1160340W	490017N1040056W	C-30
MOROCCO	274253N0125617W	355457N0005815W	C-MO
MOZAMBIQUE	264408S0300019E	102325S0404100E	C-MZ
NAURU	003517S1664956E	002723S1670130E	
NEBRASKA	305943N1040151W	430011N0951856W	C-31
NEPAL	262058N0800235E	302637N0881144E	C-NP
NETHERLANDS	504458N0032143E	533047N0071312E	C-NL
NETHERLANDS ANTILLES	120109N0691008W	122328N0681134W	C-NA
NEVADA	345912N1200014W	420021N1140255W	C-32
NEVIS	170529N0623735W	171206N0623203W	

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
NEW CALEDONIA	222400S1635846E	200504S1670229E	C-NC
NEW HAMPSHIRE	424144N0723351W	451831N0704211W	C-33
NEW JERSEY	194002S1663102E	143633S1701549E	C-NH
NEW JERSEY	385459N0753348W	412113N0735328W	C-34
NEW MEXICO	311900N1090343W	365955N1030000W	C-35
NEW YORK	402756N0794624W	450114N0715312W	C-36
NEW ZEALAND	471811S1662539E	342337S1783144E	C-NZ
NICARAGUA	104202N0874143W	150049N0830758W	C-NG
NIGER	114452N0000409E	233415N0155935E	C-NI
NIGERIA	041243N0024209E	135051N0142411E	C-NE
NIUE	190731S1700656W	185226S1700019W	
NORTH AMERICA	143527N1685556W	830609N0524503W	C-1A
NORTH ATLANTIC OCEAN	000000N0970000W	700000N0100000E	C-37
NORTH CAROLINA	335052N0841956W	363551N0752653W	C-38
NORTH DAKOTA	455618N1040249W	490006N0963316W	
NORTH KOREA	374052N1241952E	425451N1305019E	
NORTH PACIFIC OCEAN	000000N1200000E	550000N0800000W	C-3A
NORWAY	580052N0044526E	710638N0310057E	C-NO
OHIO	382351N0844952W	415830N0803048W	C-39
OKLAHOMA	333819N1030022W	365952N0942617W	C-40
OMAN	163555N0515924E	261826N0594543E	C-MU

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
OREGON	415941N1243321W	461510N1162651W	C-41
PAKISTAN	234309N0605155E	370117N0773949E	C-PK
PANAMA	071204N0830256W	093700N0770928W	C-PM
PAPUA NEW GUINEA	104103S1410514E	011846S1560407E	C-PP
PARAGUAY	272947S0624311W	191403S0542034W	C-PA
PENNSYLVANIA	394306N0803138W	421603N0744144W	C-42
PERU	181244S0811740W	000602S0684812W	C-PE
PHILIPPINES	045919N1171427E	183606N1263451E	C-RP
PITCAIRN ISLAND	250534S1301651W	243826S1245442W	C-PC
PITCAIRN ISLAND	250457S1300628W	250318S1300343W	C-PC
POLAND	490727N0140505E	545212N0240107E	C-PL
POLYNESIA	231136S1513814W	084303S1353821W	C-PL
PORTUGAL	365900N0093056W	420923N0061215W	C-P0
PORTUGUESE GUINEA	105516N0164316W	124105N0133829W	C-PU
PORTUGUESE TIMOR	102205S1233400E	090405S1250724E	C-PU
PRINCIPE	013150N0071936E	014207N0072752E	C-RQ
PUERTO RICO	175517N0671613W	183110N0653611W	C-RQ
PUERTO RICO COMMONWEALTH	175725N0671507W	183029N0654517W	C-QA
QATAR	243223N0504402E	261018N0513334E	C-QA
QUEEN ELIZABETH ISLANDS	742228N1225542W	830608N0610748W	C-QA
REUNION ISLAND	212235S0551206E	205122S0555111E	C-44
RHODE ISLAND	411903N0715045W	420101N0710741W	C-44

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	CCOUNTRY-CODE
RHODESIA	22254850251347E	15363650330423E	
ROMANIA	434039N0200935E	481711N0294148E	C-RO
RUANDA	02500450285059E	01032050305336E	C-RU
SAN MARINO	434621N0120344E	440909N0125700E	
SAO TOME	00012450062747E	002435N0064601E	
SAO TOME AND PRINCIPE	000716N0062555E	014606N0072227E	
SARDINIA	385134N0080755E	411611N0094918E	
SAUDIA ARABIA	151704N0343451E	321318N0594129E	C-SA
SENEGAL	121815N0173227U	164129N0112137U	C-SG
SEYCHELLES ISLANDS	04473050551256E	04155850555222E	C-SE
SICILY	363849N0120126E	381747N0153829E	
SIERRA LEONE	065452N0131826U	100005N0101551U	C-SL
SINAI PENINSULA	274714N0320908E	311857N0344836E	
SINGAPORE	011156N1033821E	012643N1035953E	C-SN
SOLOMON ISLANDS	12533551562220E	06360051623021E	
SOMALIA	01380250410212E	115348N0511703E	C-SO
SOUTH AFRICA	34474350162827E	22072950325251E	C-SF
SOUTH AMERICA	55220750811740U	122731N0343711U	
SOUTH ATLANTIC OCEAN	55000050700000U	000000N0250000E	C-2A
SOUTH CAROLINA	320103N0832203U	351326N0783305U	C-45
SOUTH DAKOTA	423018N1040250U	455640N0962729U	C-46
SOUTH EAST ASIA	000001N0000108E	000054N00000215E	

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
SOUTH KOREA	330853N1260821E	383024N1305829E	
SOUTH PACIFIC OCEAN	600000S1200000E	000000N0700000W	C-4A
SOUTH YEMEN	120734N0432330E	185957N0543053E	
SOUTHERN RHODESIA	223643S0250835E	153428S0325816E	C-YS
SOUTHERN YEMEN	120734N0432330E	185957N0543053E	C-UA
SOUTH-WEST AFRICA	290128S0113615E	165138S0250835E	
SOVIET UNION	351431N0193312E	814808N1694122W	C-UR
SPAIN	360015N0091559W	434706N0042035E	C-SP
SPANISH SAMARA	204443N0170658W	274015N0083927W	
SRI LANKA	055430N0792714E	095028N0815338E	C-CE
ST CHRISTOPHER	171224N0625153W	172437N0623711W	
ST CHRISTOPHER, NEVIS, ANGUILL	171721N0625338W	172614N0624502W	C-SC
ST HELENA ISLAND	160125S0054738W	155411S0053842W	C-SH
ST PIERRE AND MIQUELON IS	464449N0562359W	470812N0560839W	C-SB
ST VINCENT	130751N0611656W	132301N0610713W	C-UC
SUDAN	033654N0215333E	230624N0382703E	C-SU
SURINAM	014938N0580413W	060039N0535751W	C-NS
SVALBARD JAN MA	762403N0102748E	804532N0333904E	C-JS
SWAZILAND	271752S0304739E	254350S0320759E	C-WZ
SWEDEEN	562314N0110151E	690403N0240405E	C-SW
SUITZERLAND	454901N0056754E	474849N0102914E	C-SZ

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CODE
SYRIA	321842N0352902E	372210N0423149E	C-SY
TAIWAN	215649N1200442E	251628N1215120E	C-TW
TANZANIA	114211S0294108E	005637S0403316E	C-TZ
TENNESSEE	345825N0901902W	364109N0813920W	C-47
TEXAS	254952N1064152W	362949N0933007W	C-48
THAILAND	053845N0972126E	202627N1053830E	C-TH
TIMOR	110006S1223825E	081930S1272111E	C-TO
TOGO	060320N0001144W	110915N0015844E	C-TL
TOKELAU ISLANDS	102601S1714817W	085810S1574424W	C-TN
TONGA ISLANDS	212754S1752546W	183405S1735449W	C-TD
TRINIDAD AND TOBAGO	100244N0615300W	112041N0603045U	C-TS
TRUCIAL STATES	222939N0510058E	260426N0561723E	C-TU
TUNISIA	301938N0072133E	372027N0111942E	C-UG
TURKEY	354849N0260330E	420655N0444924E	C-TC
UGANDA	012918S0293309E	041258N0350153E	C-UK
UNITED ARAB EMERATES	222939N0510058E	260426N0561723E	C-US
UNITED KINGDOM	495912N0075836W	603853N0014849E	C-UU
UNITED KINGDOM, NORTH ICELAND	495912N0075836W	603853N0014849E	C-UU
UNITED STATES	184610N1782703W	712027N0665529W	C-UU
UPPER VOLTA	092303N0053150W	150535N0022357E	C-UU
URUGUAY	345649S0582853W	300506S0530920W	C-UU
US TRUST TERRITORY OF PACIFIC	000000N1302000E	210500N1791000E	

WINDOW

WINDOW NAME	LOWER LEFT	UPPER RIGHT	COUNTRY-CCDE
USA	245721N1244326W	492239N0665759W	
USSR	351431N0193312E	814808N1694122W	
UTAH	365857N1140315W	415959N1090331W	C-49
VENEZUELA	003848N0731916W	121151N0594805W	C-UE
VERMONT	424321N0732705W	450125N0712928W	C-50
VIETNAM	083515N1020817E	231217N1092051E	C-UM
VIRGIN ISLANDS	174038N0653447W	182444N0643327W	C-VQ
VIRGINIA	363208N0834050W	392703N0752232W	C-S1
WAKE	191645N1663511E	192051N1663947E	
WAKE ISLAND	191928N1662925E	192159N1663004E	
WASHINGTON	453336N1244326W	490001N1165336W	C-53
WEST GERMANY	471618N0055254E	550219N0134814E	
WEST VIRGINIA	371155N0823823W	403828N0774359W	C-S4
WESTERN SAHARA	204443N0170658W	274015N0083927W	C-UI
WESTERN SAMOA	140338S1724653W	132737S1712538W	C-US
WINDUARD ISLANDS	115858N0614639W	145250N0592546W	
WISCONSIN	422909N0925434W	465806N0865509W	C-55
WORLD	681055S1795941W	681055N1795941E	
WYOMING	405955N1110336W	450012N1040124W	C-56
YEMEN ARAB REPUBLIC	124124N0423706E	171811N0461721E	C-YE
YUGOSLAVIA	405255N0132404E	465404N0225657E	C-YO
ZAIRE	132925S0120959E	052232N0311554E	C-CG

WINDOW

COUNTRY-CODE

UPPER RIGHT

LOWER LEFT

WINDOW NAME

C-2A
C-2I

08100350334128E
15363650330423E

18011250220053E
22254850251347E

ZAMBIA
ZIMBABWE

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F
LOCATION NAMES

THIS PAGE INTENTIONALLY LEFT BLANK

LOCATION NAMES

LOCATION NAME	LOCATION
ALBANY	434000N0734500U
ALBUQUERQUE	350500N1064700U
AMARILLO	351100N1015000U
ANCHORAGE	611300N1495400U
ATLANTA	334500N0842300U
ATLANTIC CITY	392200N0742500U
AUSTIN	392900N1170400U
BAKER	444700N1175000U
BALTIMORE	391800N0763800U
BISMARCK	464800N1004700U
BOISE	433600N1161300U
BOSTON	422100N0710500U
BUFFALO	425500N0785000U
CALGARY	510100N1140100U
CARLSBAD	322600N1041500U
CHARLESTON SC	324700N0795600U
CHARLESTON WVA	382100N0810800U
CHARLOTTE	351400N0805000U
CHEYENNE	410900N1045200U
CHICAGO	415000N0870700U
CINNINNATI	390800N0840000U
CLEVELAND	412800N0810700U
COLUMBIA	340000N0810200U
COLUMBUS	400000N0830100U
DALLAS	324600N0964600U

LOCATION NAMES

LOCATION NAME	LOCATION
DENVER	394500N1050000U
DES MOINES	413500N0933700U
DETROIT	422000N0830300U
DUBUQUE	423100N0904000U
DULUTH	464900N0920500U
EASTPORT	445400N0670000U
EL CENTRO	323800N1153300U
EL PASO	314600N1062900U
EUGENE	440300N1230500U
FARGO	465200N0954800U
FLAGSTAFF	351300N114100U
FRESNO	364400N1194800U
GARDEN CITY	375800N1005300U
GRAND JUNCTION	390500N1033300U
GRAND RAPIDS	425800N0854000U
HAURE	483300N1034300U
HELENA	463500N1120200U
HONOLULU	211800N1575000U
HOQUIAM	465900N1235400U
HOT SPRINGS	343100N0930300U
IDAH0 FALLS	433000N1120100U
INDIANAPOLIS	394600N0801000U
JACKSON	322000N0901200U
JACKSONVILLE	302200N0814000U

LOCATION NAMES

LOCATION NAME	LOCATION
JUNEAU	581800N1342400U
KANAS CITY	390600N0943500U
LANDER	425000N1084000U
LAS VAGAS	361000N1151200U
LEWISTON	462400N1170200U
LINCOLN	405000N0964000U
LONDON	430200N0813400U
LOS ANGELES	340300N1181500U
LOUISVILLE	381500N0854600U
MANCHESTER	430000N0713000U
MEMPHIS	350900N0900300U
MIAMI	254600N0801200U
MILWAUKEE	430200N0875500U
MINNEAPOLIS	445900N0931400U
MOBILE	304200N0880300U
MONTGOMERY	322100N0861800U
MONTPELIER	441500N0723200U
MONTREAL	453000N0733500U
MOOSE JAW	503700N1053100U
NASHVILLE	361000N0864700U
NEEDLES	345000N1143600U
NELSON	493000N1171700U
NEU HAVEN	411900N0725500U
NEU ORLEANS	295700N0900400U

LOCATION NAMES

LOCATION NAME	LOCATION
NEW YORK CITY	404700N0735800U
NOGALES	312100N1105600U
NOME	642500N1653000U
NORTH PLATTE	410800N1004600U
OKLAHAMA	352600N0972800U
OTTAWA	452400N0754300U
PHILADELPHIA	395700N0751000U
PHOENIX	332900N1120400U
PIERRE	442200N1002100U
PITTSBURGH	402700N0795700U
PORT AUTHUR	483000N0891700U
PORTLAND ME	434000N0701500U
PORTLAND ORE	453100N1224100U
PROVIDENCE	415000N0712400U
QUEBEC	464900N0711100U
RALEIGH	354600N0783900U
RENO	393000N1194900U
RICHFIELD	384600N1120500U
RICHMOND	373300N0772900U
ROANOKE	371700N0795700U
SACRAMENTO	383500N1213000U
SALMON	451100N1135400U
SALT LAKE CITY	404600N1115400U
SAN ANTONIO	292300N0983300U

LOCATION NAMES

LOCATION NAME	LOCATION
SAN DIEGO	324200N1171000U
SAN FRANCISCO	371700N1222600U
SAN JUAN	183000N0661000U
SANTA FE	354100N1055700U
SAULT STE MARIE	463000N0842100U
SAVANNAH	320500N0810500U
SCRANTON	412400N0753900U
SEATTLE	473700N1222000U
SHREVEPORT	322800N0934200U
SIOUX FALLS	433300N0964400U
SITKA	571000N1351500U
SPOKANE	474000N1172600U
SPRINGFIELD ILL	394800N0393800U
SPRINGFIELD MASS	420600N0723400U
SPRINGFIELD MO	371300N0931700U
ST. JOHN	451800N0661000U
ST. LOUIS	383500N0901200U
SYRACUSE	430200N0760800U
TAMPA	275700N0822700U
TORONTO	434000N0792400U
TRINIDAD	371000N1043000U
VICTORIA	482500N1232100U
WATERTOWN	435800N0755500U
WICHITA	374300N0971700U

LOCATION NAMES

LOCATION

LOCATION NAME

341400N0775700U
495400N0970700U

WILMINGTON
WINNIPEG

APPENDIX G
WIS WORKSTATION (WWS) GIPSY TERMINAL OPERATIONS

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX L. WIS WORKSTATION (WWS) GIPSY TERMINAL OPERATIONS (VIPTEK)

L.1 Introduction.

The WIS Workstation (WWS) IBM PC 3270 color terminal can be used with GIPSY as an eight_color graphics terminal. To make use of the color graphics capabilities, the terminal must meet the following requirements:

- o Be connected to the H6000 as a VIP 7705 and successfully executing BIS 7705 or ETC software.
- o Be configured with a copy of the VIPTEK software on the master catalogue as VIPTEK.EXE.
- o Have GIPSYmate system software installed on the WWS.

If these requirements are met, the user can successfully use the PC 3270 as a graphics terminal with GIPSY.

L.1.1 BIS7705. After selection from the initial start up menu, a VIP 7705 emulator program is loaded into memory. This emulator is the BIS7705. The BIS7705 program automatically loads GIPSYmate into memory as a child process. From this point the user may activate GIPSYmate either by logging onto the host and entering GIPSY through the BIS7705 emulator, or by simply pressing both SHIFT keys simultaneously to enter the GIPSYmate mode without logging onto the host.

L.1.2 ETC. After selection from the initial start up menu, a secondary menu will appear. From this menu the user may select "H6000 with Graphics", "H6000 without Graphics", or "ETC configuration". The user should select "H6000 with Graphics". After this selection a VIP 7705 emulator program will be loaded into memory. This emulator is ETC. This ETC program will load GIPSYmate into memory as a child process. From this point the user may activate GIPSYmate either by logging onto the host and entering GIPSY through the ETC emulator, or by simply pressing both SHIFT keys simultaneously to enter the GIPSYmate mode without logging on to the host.

L.2 Operations.

The following steps should be followed to use GIPSY with the WWS color terminal:

- 1) Turn on your WWS and follow the instructions above for the appropriate emulator package you have installed on your system.
- 2) Enter "GIPSY" at the system level prompt to initiate GIPSY execution. If the terminal is defined to GIPSY, the initialization procedures in GIPSY will configure GIPSY to the specifications of WWS (Colors, screen height and width, etc.). If their terminal is not defined to GIPSY, the user will be prompted

to input a code defining the terminal type. Figure L-1 shows the GIPSY prompt when the terminal has not been previously defined in the system definition file, (..SYSDEF). After GIPSY starts up and determines the terminal type, it will automatically cause the VIP 7705 emulator to turn control over to VIPTEK--the child processor.

- 3) Exiting GIPSY by entering the DONE command will switch control from the TEKEM back to the VIP 7705 emulator.

*GIPSY
This device is not defined in the GIPSY System Definition File.
Please enter appropriate GIPSY device code:
1-TEK 4014, 2-VIP 7705, 3-VIP 786, 4-IBM 2741, 5-KSR 33 TTY
6-TEK 4012, 7-TEK 4051, 8-TEK 4014(INTELLIGENT), 9-TEK 4027
10-BATCH, 11-TEK 4014(WIN), 12-UNIVAC 1652(NIDS), 13-TEK 4107
14-WSGT, 15-TEK 4054, 16-WSGT(SYNC), 17-WSGT(PHASE-III),
18-WIS WORKSTATION
DEVICE CODE =18

Figure G-1. GIPSY Undefined Terminal Prompt

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H
INDEX

\$BODY-LINE	6-21
\$BODY-PART	6-21
\$DATE	3-48
\$ISPPTR	3-48
\$LINES-LEFT	6-22
\$PAGE-NUMBER	6-22
\$PRINT-TIME	6-23
\$PRINT-DATE	6-22
\$TIME	3-48
\$TOTAL-PAGES	6-23
ACCESS	3-141
ACCUMULATE	3-122
ADD	3-154
Adding text	3-133
AFT.	3-29
APPEND	3-120, 3-122
Arithmetic operators	3-45, 3-143
Arithmetic expressions	3-45
ASSIGN	3-143
AUTO	3-126
AXIS	3-126
Bar graph	2-12, 3-81
Batch job	3-54
Binary Integer	3-33.1
Binary Floating Point	3-33.1
Body table	6-4
BOOK, direct	6-37
BOOK, display	6-24
BOOK file	6-23
BOOK review commands	6-30
BREAK command	3-67
Browsing	4-1
BUILD FDT	3-36
BUILD output	6-23
Calculate data	3-61, 3-65
Catalog-file descriptors	3-6
Category and Section definition	3-66
Category	3-79
//CATEGORY	3-23
CHANGE	3-149
CHANGES	3-38
Character size	3-9
Circles, adding	3-193
//CIRCLES	3-23
Classification caveats	6-38
Classification codes	3-10
Classification markings	3-9
CLEAR	3-27
//CLEAR PAGE	3-23
Color processing	3-27

COLOR	3-86
//COLS	3-23
Column definition	3-59
Comments	3-26
COMPLETE	3-38
Composite Expression	3-43.1, 3-47
Concatenation	3-48
Conditional assignments	3-47
Conditional expressions	3-37
Conditional expressions, predefined	3-41
//CONTINUE	3-23
Comparison operators	2-7
Coordinates	3-48
//COPY	3-23
CUELIB	3-7
Current file	3-12
Curve graph	3-99
DAFC	1-6, 3-7
DAFC, save	3-21
DAFC, recall	3-20
Data file	3-29
Data manipulation	6-1
Data modification	3-46, 4-49, 4-4
Data retrieval	3-43, 4-1
Data selection	3-45
Data types	3-4, 3-32
Data, user input	3-28
DATSEL	3-9
DEFAULT	3-123
DEFINE	3-153
Define charater.	3-179
Define fieldname	3-32.1
Define process	3-49
DEFINE TERMINAL	3-16
DELETE	3-146
DELETE field definitions	3-35
Density	3-85
DIF, save	3-22
DIFFERENCE	3-122
DIRECT BOOK	6-37
DISPLA	3-9
DISPLAY	3-158, 3-190
DISPLAY BAR	3-81, 3-86
Display Book, interactive	6-24
Display Book, Non-interactive	6-25
Display control	3-171
DISPLAY CURVE.	3-99
DISPLAY GANTT	3-107
Display, geographic	3-157
DISPLAY LINE GRAPH	3-97, 3-99

DISPLAY MAP	3-159
Display map on slide	5-1
DISPLAY PIE	3-111
DISPLAY POINT	3-97
DISPLAY REPORT	3-69
DISPLAY SYMBOL	3-132
DISTANCE	3-194
DO PROCESS	3-49
DONE statement	3-27, 3-55
Download data	5-1
//ECHO	3-23, 3-53
EJECT statement	6-20
Error correction	3-1, 3-5, 3-12
EXECUTE	3-27
EXPLICIT	3-60, 3-64, 3-77
EXTRACT	3-198
FDT	1-6, 2-6, 3-32
FDT, save	3-18
//FDT	3-24, 4-6
Field assignment	4-5
Field definition, add	6-2
Field definition, delete	3-35
Field definition, insert	3-35
Field, extened	3-4
Field manipulation	6-1
FIELD TABLE	3-47, 3-49, 4-3, 4-5
FILE	3-8
File Structure Table (FST)	1-6, 3-30, 3-30.1
File types	2-6
FILL	3-86, 3-124
FINAL	6-10
FIXDEC	3-71, 3-123
FLOAT labels	3-113
FOR INITIAL	3-47
Gantt chart	3-107
GCFILE	3-8
GDR	3-9, 6-1
GDS	1-6, 2-9, 3-8, 3-69
GDS, access	3-19
GDS, save	3-19
//GDT	3-24
GENERATE	3-132, 3-190
Geographic display	3-157
GEOMOD	3-9
GET IDSII STRUCTURE	3-30
GFRC file	3-29
GIPSY batch	3-54
GIPSYD	3-7
GIPSYG	3-7, 3-188
GIPSY initialization	3-6

GIPSY language	2-9, 3-11
GIPSY modes	2-9, 3-200
GIPSY outputs	2-12
GIPSY prompt	2-9, 3-1
GIPSY release number	3-1
GIPSYR	3-7, 6-1
GIPSY statements	3-1, 3-11
GIPSY, terminating	3-27
GLOBAL	3-37
Graphics off	3-135
GREAT CIRCLE	3-185
GRID	2-23, 3-126, 3-127, 3-173
GROUP definition	6-23
Header table	6-4
Histogram	2-12, 3-91
//HUH	3-24
I-D-S/II	1-7, 3-29
//IDSII STRUCTURE	3-24
//IDT	3-24
INCLUDE	3-22, 3-44
Index file, describing FDT	3-36
Index file	1-7, 3-29
INITIAL	6-10
INPUT	3-154
INPUT statement	3-156
INSERT field definitions	3-35
INSIDE	3-113
Integrated file	1-7
Interrupt statements	3-23
ISP file	3-29
JDAC	3-28
Labels	3-113
//LAST	3-24
LIBRARY statement	3-28
LIMIT	3-118, 3-145, 3-146, 3-197
LIMIT SYMBOLS	3-132, 3-197
//LIMIT	3-24, 3-120
Limiting decimal places	3-123
Line color	3-99
Line graph	2-12, 3-91
Line numbers	3-12
LINE options	3-98
LINE statement	6-10, 6-11
Line types	3-106, 3-166
LIST, data	3-198
//LIST	3-24
//LOCATION	3-25
LOCATION	3-194
LOCATION TABLE	3-171
LOCK	3-125, 3-130

Logic table	1-7, 3-42
//LOGIC	3-25
MAGIC/GIPSY interface	5-1
Map details	3-164
Map, display on slide	5-1
Map file	3-160
MAP definition	3-158
Math table	1-7, 3-46, 3-48
//MATH	3-25
Matrix generation	3-56
Matrix modification	3-116
MAXDEC	3-71, 3-123
Mode	1-7
Module	3-8
MTXGEN	3-9
NAMED	3-193
NO-MSG	3-8
//NOTE	3-25
Output table	6-4
Output table routines	6-21
Packed Decimal	3-33
PAGING	3-91, 3-103
Partial field notation	1-7, 3-4, 3-34, 4-6
Parts definition	6-4
//PAUSE	3-25
PCS	1-8, 3-8, 3-11
PCS, save	3-19
Picture processing	3-200
Picture recall	3-201
Picture save	3-200
//PICTURE	3-25, 3-202
Pie chart	2-12, 3-111
Plot picture	3-201
Point graph	2-12, 3-91
Point options	3-98
POST-DATSEL	3-9
POST-DISPLA	3-9
POST-GEOMOD	3-9
POST-MTXGEN	3-9
POST-SYNTAX	3-9
PRE-DATSEL	3-9
PRE-DISPLA	3-9
PRE-GEOMOD	3-9
PRE-MOD	3-9
PRE-MTXGEN	3-9
PRE-SYNTAX	3-9
//PREVIEW	3-25
Print	4-1, 4-2
Print to disk file	4-3
//PROCESS	3-25

PROJECTION	3-159, 3-168
Prompt	1-8
Prompt character	3-14
Prompt, user input	3-56
Proportion specifications	3-125
//PURGE	3-25
QDF	1-8, 2-7, 3-8, 4-1, 4-3, 4-9, 4-10, 6-2
QDF manipulation	6-3
QDT, save	3-19
QDT	1-8
//QDT	3-25
//QLT	3-25
QUALIFY	3-37, 3-199, 4-1, 6-3
Random Access	1-8
Range process	3-63
RANGE	3-60, 3-73
RECNBR	3-43.1
Record output table	4-9
Record_type	3-32, 3-43.1, 3-47.1
Refresh	3-135
Relational operators	3-38
Relative	1-8
Relative Expression	3-43.1, 3-47.1
RENAME	3-147
//REPORT	3-25
Report Building	6-4
REPORT, build new	3-141
Report, build tabular	3-59
Report definition	6-4
Report, display	3-67, 3-69
Reports, modifying	2-12, 3-116
Reports, multiple	3-66
Report, review	3-143
REPORT, save	3-141
Reports, statistical	3-56
Report, tabular	2-12, 3-69
Report totals	3-120
Reports, types	3-69
RESORT	6-3
RETAIN picture	3-200
RETRIEVE	3-43.1, 3-47, 3-67, 3-199, 4-1, 4-6
RETRIEVE from index	3-44
RETURN	3-12, 3-14
//RETURN	3-14, 3-25
REVIEW	3-157
//ROUTINES	3-26
Row and Column definition	3-59
//ROWS	3-26
RUN command	3-44, 3-188
SAVE, DAFC	3-20

SAVE, data fields	2-22
SAVE, DIF.	3-22
SAVE, GDS.	3-19
SAVE, FDT.	3-18
SAVE, PCS.	3-19
SAVE, QDF.	3-19
Section Definition	3-66, 3-79
//SECTIONS	3-26
Security	1-9
SELECT	3-59, 3-62, 3-75
Sequential Access	1-8
SET CLEAR	3-14
SET COLOR	3-27
SET COPY	3-14
SET DATE	3-14
SET DTG	3-14
SET ECHO	3-14
SET FONT	3-15
SET GRAPHICS	3-14
SET LINE COLOR	3-99
SET MAP	3-159, 3-171
SET MESSAGES	3-14
SET MONITOR	3-45
SET OUTPUT	4-4
SET PROTECT	3-171
SET RECORDING	3-14
SET SIZE	3-15
SET VISIBLE	3-171
SET WAIT	3-14
SHADE	3-85, 3-124
SHOW	3-103, 3-106
SIZE options	3-98
SORT	3-36, 4-4, 6-2
SPACE statement	6-19
STACKING	3-82, 3-86, 3-91, 3-98
Statement composition	3-3
Step Graph	3-103, 3-107
Subroutines	3-28
SUBSET	3-148
Subsetting	4-3
Symbol plot	3-174
SYMBOL TABLE	3-175
Symbols	3-132
Symbols, adding	3-132, 3-193
Symbols, limit	3-132, 3-197
Symbols, user defined	3-179
//SYMBOLS	3-26
SYNTAX	3-9
Syntax error correction.	3-5
Temporary PCS	3-12

Terminal type	3-17
Terminal types supported	2-3
Terminating GIPSY	3-27
Text, adding	3-198
TEXT TABLE	3-133
//TEXT	3-26
Tic marks	3-126, 3-127
Title	3-9, 3-86
Title, modifying	3-11
TOPBOT	3-10
Track options	3-184
TRACK TABLE	3-184
Tracks, adding	3-190
//TRACKS	3-26
Trailer table	6-10
TRANSFER	3-188, 3-200
//TRMDEP	3-17
TSS logon	3-1
TSS command	3-55
TSS commands	3-26
Type Expression	3-43.1, 3.47.1
UFAS	1-8
UFAS file	3-29
UNLOCK	3-125
USE	3-59, 3-72
USE process	3-60
User input	3-56
User programs	3-27
USING	3-98, 3-101
VALUE	3-81
Vector mode	3-85
WEDGE	3-113
WINDOW	3-159, 3-166
//WINDOW	3-26
WINDOW TABLE	3-168
WIS/CUC	5-1
WITHIN coordinate	3-38
WORLD2	3-160
X-TITLE	3-97
X-axis	3-126
Y-TITLE	3-97
Y-axis	3-126
Z-248	5-1
ZOOM	3-195

THIS PAGE INTENTIONALLY LEFT BLANK

DISTRIBUTION

Addressees	Copies
DSSO Codes	
JPAM-P (Record and Reference Set)	3
JNSL	2
JNCR	3
JNCE	6
JNON	3
JNGD	6
JNGG	25
JWS.	4
JWO.	2
JCO.	2
JCRO	1
ADP Support Group, COC/Systems Software	
SHAPE APO New York 09055.	12
AFCENT/SSC, ATTN: DSO SFC MARTIN, USAE Box F	
AP0 AE 09703-5000.	1
Alaskan Air Command, ATTN: SCOS	
Elmendorf AFB, AK 99506-5001	1
ANMCCOD	
Ft. Ritchie, MD 21719-5010	1
ATC-ACDSD, Data Automation, ATTN: Systems Division	
Randolph AFB, TX 78148	1
Air Force Combat Operations Center, AFCOS/X000, Room BD967,	
The Pentagon, Washington, DC 20301-7010	2
7CG DOWHW, Room 1D1031, The Pentagon, Attn: Chuck Elmore	
The Pentagon, Washington, DC 20301	7

Addressees	Copies
Air Force Data Services Center, IRS, Room 1C536, The Pentagon, Washington, DC 20301	1
Air Force Data Systems Design Center Gunter AFB, AL 36114	5
CCSA, ATTN: MOCS-C, Room MF741C, The Pentagon, Washington, DC 20301	2
CCSA-E, ATTN: ASQE-CC-SS APO New York, NY 09403-0136.	1
CINCUSAREUR, Heidelberg (AEAGC-DPCE) APO New York 09403	1
Commander, MSC Atlantic, Military Ocean Terminal, Bldg 42 Attn: L-6, WIN Site, Bayonne, NJ 07002-5399	1
USCINCLANT, Code J633C ATTN: Technical Library, Norfolk, VA 23511	5
Commanding Officer, LANTCOM, ATTN: J3 Norfolk, VA 23511-5105	1
Commanding Officer, LANTCOM, ATTN: J07401 Norfolk, VA 23511-5105	1
Commanding Officer, LANTCOM, Code 34, Bldg. NH 95 Norfolk, VA 23511-5105	5
8th US Army, ATTN: EACJ-FD-U APO San Francisco, CA 96301-0009	1
22 AF/DOC Travis AFB, CA 94535	2
4602 CPUSS/ADBRHP Peterson AFB, CO 80914	1
Defense Technical Information Center, Cameron Station Alexandria, VA 22314	2
Det 2, HQ AWS/XOOOX, Weather Support Division, Room BD927, The Pentagon, Washington, DC 20330	1

Addressees

Copies

Director for Command, Control, and Communications Systems, J-6, The Joint Staff, ATTN: Chief, National Military Command Centers Division (J-6C), The Pentagon, Washington, D.C. 20318-3000.	1
Director for Operations, J-3, The Joint Staff, ATTN: Chief, Command Systems Operations Branch (CSOB), The Pentagon, Washington, DC 20318-3000	1
Director USAWC Operations Group, Box 315 Carlisle Barracks, PA 17013-5050	2
FOCCEUR, NAVEUR, Box 12 FPO New York, NY 09510-3400	1
HQ AAC, ATTN: J5 Elmendorf AFB, AK 99506	1
HQ ADCOM, ATTN: J3X Peterson AFB, CO 80914	1
HQ AF Systems Command, HQ AFSC/SCRZ Andrews AFB, MD 20334-5000	2
AFWMPRT/SC Ft. Ritchie, MD 21719	1
HQ AF, Systems Command, HQ AFSC/MPME Andrews AFB, MD 20334-5000	1
HQ AF, AFDFDC/PREJ Gunter AFB, AL 36114	1
HQ AF RES/DOOR Robins AFB, GA 31098	1
HQ AF, SOC/XPX Hurlburt Field, FL 32544-5000	1
HQ AF, Air Force Personnel Analysis, AFMPXAR, Room 5C360, The Pentagon, Washington, DC 20330	1
HQ MAC, 1500 CSGP/STSW Scott AFB, IL 62225-5431	2
HQ ATC, 3390 TCHTG/TTMKPQ Keesler AFB, MS 39534-5000	4

Addressees	Copies
HQ Air Force Manpower Center, AFMPC/DPMDMS, Bldg. 499 Randolph AFB, TX 78150-6001	1
HQ Air University, ATTN: AU/SCO Maxwell AFB, AL 36112-5000	1
HQ Air University, ATTN: AWC/PG Maxwell AFB, AL 36116	1
HQ FORSCOM, WWMCCS Division, DCSOPS, Bldg. 101 Ft. Gillem, Forest Park, GA 30050	2
HQ FORSCOM, ATTN: AFOP-OC Ft. McPherson, GA 30330-6000	1
HQ FORSCOM, FCJ6-WWS, Bldg. 200 Ft. McPherson, GA 30330-6000	1
HQ MAC/SCDS Scott AFB, IL 62225-5431	1
HQ MAC/XOOX Scott AFB, IL 62225	1
HQ MAC/XOOXA Scott AFB, IL 62225-6001	1
HQ PACAF/SCSW Hickam AFB, HI 96853-6343	2
HQ PACAF/DOXO Hickam AFB, HI 96853-6343	1
HQ SAC/DOCFS Offut AFB, NE 68113-5000	1
HQ SAC/SCCS Offut AFB, NE 68113-5000	1
HQ SAC/SCOSFWC Offut AFB, NE 68113-5001	2
HQ SOUTHCOM, Code SCJ6-ASD APO Miami, FL 34003-0226	5
1912th CSGP/DOR Langley AFB, VA 23665-6345	2

Addressees	Copies
HQ TAC/DOCS Langley AFB, VA 23665-6345	3
HQ USCINCPAC-D, J66213, Box 32A Camp Smith, HI 96861-5025	1
HQ USEUCOM, ATTN: ECDSC-XS APO New York, NY 09128-4209	2
HQ USEUCOM, ATTN: ECJ3-OD APO New York, NY 09128-4209	1
HQ US Force Korea, J6 JCIS- ACTIVITY, J6-JCIS-WA APO San Francisco, CA 96301-0010	2
HQ US Force Japan, ATTN: AJGC-P APO San Francisco, CA 96320	1
HQ USEUCOM, ATTN: ECJ3-OD-R APO New York, NY 09128-4209	1
HQ USMC, CMC Code POR-14 Washington, DC 20380	1
HQ USSOUTHCOM/J6-S APO Miami, FL 34003	1
INSCOM-AISD-IDHS, (GIPD-Systems) Ft. Bragg, NC 28307	1
MSCPAC, Attn: P-811. WJN Site Oakland, CA 94625-5010	1
MTMC, ATTN: MT-PLO Washington, DC 20315	1
MTMC, ATTN: MT-PLS Washington, DC 20315	1
MTMC, MT-IMC-S, Room 428, 5611 Columbia Pike Falls Church, VA 22041-5050	1
MTMCEA, ATTN: MTE-PL Bayonne, NJ 07002	1
MTMCWA, ATTN: MTW-PL Oakland, CA 94626	1

Addressees	Copies
NCTS, Washington Navy Yard, Bldg. 196, ATTN: Code 50-P Washington, DC 20374-5069	4
Naval Electronic Systems Command, NAVELEX PAX, PD 6311B, Bldg. 1363, NAS Patuxent River, MD 20670	2
NGB-ARR, Room 1C663, Attn: Maj. JAMISON, The Pentagon, Washington, DC 20310-2500	1
OJCS/SAGA, Room 1D936, The Pentagon, Washington, DC 20301	7
OO-ALC/XPO Hill AFB, UT 84056-5990	7
PACOPSUPPFAC, Code OLS, Box 9 Pearl Harbor, HI 96860-7150	1
Puget Sound Naval Shipyard, Code 114 Bremerton, WA 98314-5000	5
US Air Force 7th Communications Group/DOWW Washington, DC 20330-6345	1
US Air Forces, Europe, HQ USAFE/SCIWSO, APO New York, NY 09094-5001.	1
US Army Pacific, ATTN: ADCSOPS(APOP-OPO) Ft Shafter, HI 96858	1
US European Command, ATTN: ECJ6-COD APO New York, NY 09128-4209.	1
US Pacific Command, ATTN: J66221, Bldg 35, Room 336 Camp Smith, HI 96861-5025.	1
US Southern Command, ATTN: SCJ6-A APO Miami, FL 34003-5000	1
USAISC-MTMC, ATTN: MODS Team, Room 526 Falls Church, VA 22041-5050	1
USCENTCOM, CSSE-DEW MacDill AFB, FL 33608-7001	1
USSOCOM, ATTN: SOJ6-CS, Bldg 501 MacDill AFB, FL 33608	1

Addressees

Copies

USTRANSCOM, ATTN: TCJ6-SS

Scott AFB, IL 62225 1

USTRANSCOM/TCJ3/4JT

Scott AFB, IL 62225-7001 1

WESTCOM, ATTN: APOP-OPP

Ft. Shafter, HI 96858 1

216

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 1 February 1991	3. REPORT TYPE AND DATES COVERED Users Manual		
4. TITLE AND SUBTITLE Graphic Information Presentation System (GIPSY) Users Manual		5. FUNDING NUMBERS		
6. AUTHOR(S) Various				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Joint Data Systems Support Center (JNGG) BF670, The Pentagon Washington, D.C. 20301-7010		8. PERFORMING ORGANIZATION REPORT NUMBER UM 7-91		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Communications Agency Washington, D.C. 20301-7010		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The Graphic Information Presentation System (GIPSY) is a general purpose graphics and information display capability. It combines the tools of data retrieval, information processing, formatted reports, tabular, graphic, and geographic display into a single integrated on-line interactive system. It is a file and data independent system that is driven by a high level user oriented language. The graphic display capabilities were implemented using a device independent approach which allow the single integrated system to support multiple dissimilar devices. GIPSY automatically reconfigures itself to the capabilities and unique requirements of the terminal to which the user is logged onto. The graphic display capabilities were the primary basis for the initiation of this system. However, GIPSY very effectively serves as an information handling system to connect the users data base to a large set of on-line interactive query and display capabilities.				
14. SUBJECT TERMS			15. NUMBER OF PAGES 426	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT	